Final Report for AOARD Grant FA2386-10-1-4052

"Mass estimation and its applications"

Date: 23 Feb 2012

Name of Principal Investigators:

- e-mail address : kaiming.ting@monash.edu
- Institution : Monash University
- Mailing Address : Gippsland School of Information Technology, Monash University, Victoria 3842 Australia.
- Phone : +61 3 990 26241
- Fax : +61 3 99026879

Period of Performance: 01/March/2010 – 01/March/2012

Abstract:

In the first year of this two-year project, we had established that the new modeling mechanism---mass estimation---has a strong theoretical underpinning for prediction and data modeling. We had also determined that mass–based approaches have time and space complexities more favorable than existing approaches in a number of data mining tasks e.g., anomaly detection, clustering and information retrieval.

In the second year, we had developed (i) a new density estimator based on mass, and (ii) a new generative classifier based on mass.

This project has produced a total of four publications in top conferences in the field. In addition, two journal papers and one conference paper are currently under review. This outcome is more than double the number of publications we had specified in the project proposal two years ago.

Introduction:

This project had planned to deliver a fundamentally different approach to design data mining algorithms; instead of relying on density estimation, a new method called mass estimation was introduced. The new method overcame the two key limitations of existing data mining algorithms which (a) require large data size in order to build a good performing model; and (b) they are restricted to low dimensional data sets because of high processing time and memory requirements. This project had aimed to

1. Establish the theoretical underpinning of mass estimation as a computational model for prediction and data modelling.
2. Develop a formalism in which mass estimation can be applied to various different tasks.
3. Provide evidence that mass estimation is a practical approach that is more efficient and effective than many existing approaches in real applications.

Experiment: N/A

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
22 FEB 2012	**Final**	**01-03-2010 to 01-03-2012**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Mass estimation and its applications	**FA23861014052**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
Kai Ming Ting	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Gippsland School of Information Technology, Monash University,Gippsland Campus, Churchill,Victoria 3842,Victoria 3842,AU,3842	**N/A**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
AOARD, UNIT 45002, APO, AP, 96338-5002	**AOARD**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
	AOARD-104052

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This project established that the new modeling mechanism "mass estimation" has a strong theoretical underpinning for prediction and data modeling. It showed that mass-based approaches have time and space complexities more favorable than existing approaches in a number of data mining tasks e.g., anomaly detection, clustering and information retrieval, and developed (i) a new density estimator based on mass, and (ii) a new generative classifier based on mass. The results have been published in top conferences and accepted for top journals.

15. SUBJECT TERMS
Classification, Clustering, Regression, Anomaly Detection, Data Mining

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**112**	

Results and Discussion:

In the first half of this project from February 2010 to February 2011, we had established the theoretical underpinning of mass estimation, developed the first mass-based formalism, and provided some preliminary evidence that the new approach is more efficient and effective than existing density-based, distance-based and SVM approaches in regression, anomaly detection, information retrieval and clustering [1,2].

In the second half of this project from March 2011 to February 2012, we had focused on two directions. First, we showed that existing density-based approaches can be significantly improved, especially in terms of runtime, by using a new density estimation method [3]. This sets a new benchmark for density-based algorithms. The new density estimation method is constructed from mass since mass is a more fundamental measure than density, and it had been shown to be more efficient than existing density estimation methods. Second, we investigated a mass-based approach to build a generative classifier and it is found to be either competitive with or better than existing Bayesian classifiers, in term of predictive accuracy [6]. The new generative classifier also has better time complexity than existing Bayesian classifiers.

In addition to the above two foci, we had also completed the following during the second half of this project:

- Published a paper on mass-based anomaly detection in data streams [4].
- Submitted a journal version [7] of KDD 2010 paper [1] to Machine Learning Journal.
- Submitted a journal version [8] of IEEE ICDM 2011 paper. We were invited to submit this paper to International Journal of Knowledge and Information Systems because the IEEE ICDM 2011 paper [3] was one of the few papers selected to be considered for the best paper award. Our paper was one of the 101 accepted regular papers from over 800 submissions.

Though not directly supported by this grant, the work produced in this project has directly contributed to an application of mass in information retrieval, reported by Zhou, Ting, Liu and Yin [5]. Our earlier work [1] showed that transforming the feature space to a mass space, and then applying an existing information retrieval system in the mass space produced better retrieval results than the same system operated in the original feature space. This is an indirect application of mass. The latest work [5] proposed a new information system that solves the problem directly using mass. The new system further improved the retrieval performance achieved by existing systems in the mass space.

Attachments: The two 2011 publications and the papers currently under review are attached. The 2010 publications had been submitted last year.

List of Publications, resulted from this project:

[1] Kai Ming Ting, Guang-Tong Zhou, Fei Tony Liu & Tan Swee Chuan. (2010). Mass Estimation and Its Applications. *Proceedings of The 16th ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2010*. pp. 989-998.

[2] Kai Ming Ting & Jonathan R. Wells. (2010). Multi-Dimensional Mass Estimation and Mass-based Clustering. *Proceedings of The 10th IEEE International Conference on Data Mining*. pp.511-520.

[3] Kai Ming Ting, Takashi Washio, Jonathan R. Wells & Fei Tony Liu. (2011). Density Estimation based on Mass. *Proceedings of The 11th IEEE International Conference on Data Mining*. pp. 715-724.

[4] Swee Chuan Tan, Kai Ming Ting & Fei Tony Liu. (2011). Fast Anomaly Detection for Streaming Data. *Proceedings of the International Joint Conference on Artificial Intelligence 2011*. pp.1151-1156.

Related publication which is not supported by Grant FA2386-10-1-4052:

[5] Guang-Tong Zhou, Kai Ming Ting, Fei Tony Liu, Yilong Yin. (2012). Relevance Feature Mapping for Content-based Multimedia Information Retrieval. *Pattern Recognition*. Vol.45: 1707-1720.

Papers currently submitted for review:

[6] Sunil Aryal, Kai Ming Ting & Jonathan R. Wells. MassCfier: A new generative classifier based on Mass. Submitted to *ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2012*.

[7] Kai Ming Ting, Guang-Tong Zhou, Fei Tony Liu & Tan Swee Chuan. Mass Estimation. Submitted to *Machine Learning Journal*.

[8] Kai Ming Ting, Takashi Washio, Jonathan R. Wells & Fei Tony Liu. Density Estimation based on Mass. Submitted *to International Journal of Knowledge and Information Systems*.

Mass Estimation and Its Applications

Kai Ming Ting, Guang-Tong Zhou*, Fei Tony Liu, James Tan Swee Chuan†
Gippsland School of Information Technology
Monash University
Australia
{kaiming.ting,tony.liu}@monash.edu, zhouguangtong@gmail.com,
jamestansc@unisim.edu.sg

ABSTRACT

This paper introduces mass estimation—a base modelling mechanism in data mining. It provides the theoretical basis of mass and an efficient method to estimate mass. We show that it solves problems very effectively in tasks such as information retrieval, regression and anomaly detection. The models, which use mass in these three tasks, perform at least as good as and often better than a total of eight state-of-the-art methods in terms of task-specific performance measures. In addition, mass estimation has constant time and space complexities.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Miscellaneous; I.5 [**Pattern Recognition**]: General

General Terms

Algorithms, Theory

1. INTRODUCTION

> 'Estimation of densities is a universal problem of statistics (knowing the densities one can solve various problems.)' — V.N. Vapnik [16].

Density estimation has been the base modelling mechanism used in many techniques designed for tasks such as classification, clustering, anomaly detection and information retrieval. For example in classification, density estimation is employed to estimate class-conditional density function (or likelihood function) $p(x|j)$ or posterior probability $p(j|x)$—the principal function underlying many classification methods e.g., mixture models, Bayesian networks, and Naive Bayes. Examples of density estimation include kernel density estimation, k-nearest neighbours density estimation, maximum likelihood procedures or Bayesian methods.

We shows in this paper that a new base modelling mechanism called **mass estimation** possesses different properties from those offered by density estimation:

- A mass distribution stipulates an ordering from core points to fringe points in a data cloud. In addition, this ordering accentuates the fringe points with a concave function—fringe points have markedly smaller mass than points close to the core points. These are the fundamental properties required for many tasks, including anomaly detection and information retrieval. In contrast, density estimation is not designed to provide an ordering.
- Mass estimation is more efficient than density estimation because mass is computed by simple counting and it requires only a small sample through an ensemble approach. Density estimation (often used to estimate $p(x|j)$ and $p(j|x)$) require a large sample size in order to have a good estimation and is computationally expensive in terms of time and space complexities [7].
- Mass can be interpreted as a measure of relevance with respect to the concept underlying the data, i.e., core points indicate that they are highly relevant and fringe points indicates that they are less relevance. We demonstrate in this paper that a relevance feature space consists of a vector of masses estimated from data is very effective for three data mining tasks: information retrieval, regression and anomaly detection.

Mass estimation has two advantages in relation to efficacy and efficiency. First, the concavity property mentioned above ensures that fringe points are 'stretched' to be farther from the core points in a mass space—making it easier to separate fringe points from those points close to core points. This property, otherwise hidden, can then be exploited by a data mining algorithm to achieve a better result for the intended task than the one without it. We show the efficacy of mass in improving the task-specific performance of four existing state-of-the-art algorithms in information retrieval and regression tasks in this paper. The significant improvements are achieved through a simple mapping from the original space to a mass space using the mass estimation mechanism introduced here.

*Guang-Tong is a student at School of Computer Science and Technology, Shandong University, China. He was visiting Monash University, supported by a scholarship from Shandong University, when this research was conducted.

†James is now at SIM University, Singapore.

KDD'10, July 25–28, 2010, Washington, DC, USA.

Table 1: Symbols and notations.

$\mathcal{R}^u$	A real domain of u dimensions
x	An one-dimensional instance in $\mathcal{R}$
$\mathbf{x}$	An instance in $\mathcal{R}^u$
D	A data set of $\mathbf{x}$, where $\|D\| = n$
$\mathcal{D}$	A subset of D, where $\|\mathcal{D}\| = \psi$
$\mathbf{z}$	An instance in $\mathcal{R}^t$
D'	A data set of $\mathbf{z}$
h	Level of mass distribution
t	Number of mass distributions in $\widetilde{\mathbf{mass}}(.)$
$m_i(.)$	Mass base function defined using binary split s_i
$mass(.)$	Mass function which returns a real value
$\widetilde{\mathbf{mass}}(.)$	Mass function which returns a vector of t values

Second, mass estimation offers to solve a problem more efficiently using the ordering derived from data directly—without distance or related expensive calculation—when the problem demands ranking. An example of inefficient application is in anomaly detection tasks where many methods have employed distance or density—a computationally expensive process—to provide the required ranking. An existing state-of-the-art density-based anomaly detector `LOF` [4] (which has quadratic time complexity) cannot complete a job involving half a million data points in less than two weeks; yet the mass-based anomaly detector we have introduced here completes it in less than 40 seconds! Section 4.3 provides the detail of this example.

Section 2 introduces mass and mass estimation, together with their theoretical properties. We also describe an efficient method to estimate mass in practice. Section 3 describes a mass-based formalism which serves as a basis of applying mass to different data mining tasks. We present a realisation of the formalism in three different tasks: information retrieval, regression and anomaly detection, and report the empirical evaluation results in Section 4. The relation to kernel density estimation is given in Section 5. We provide related work, the conclusions and future work in the last two sections.

2. MASS AND MASS ESTIMATION

Data mass or **mass** is defined as the number of points in a region; and two groups of data can have the same mass regardless of the characteristics of the regions (e.g., density, shape or volume). Mass in a given region is defined by a rectangular function which has the same value for the entire region in which the mass is measured.

Identifying a region occupied by a group of data in itself is a clustering problem, but mass can nonetheless be estimated without clustering. We show in this section that mass can be estimated in a way similar to kernel density estimation without involving clustering at all by using a function similar to a kernel function.

Note that **mass is not a probability mass function, and it does not provide probability**, as probability density function does through integration.

The detail of mass estimation is provided in the following two subsections. In Section 2.1, we show how to estimate a mass distribution for a given data set, and the theoretical properties of mass estimation. Section 2.2 describes an approximation to the theoretical mass estimation which works more efficiently in practice. This paper focuses on one-dimensional mass distribution only. The symbols and notations used are provided in Table 1.

2.1 Mass distribution estimation

We first show level-1 mass distribution estimation in Section 2.1.1. We then generalise the treatment for high level mass estimation in Section 2.1.2.

2.1.1 Level-1 mass distribution estimation

Here, we employ a binary split to divide the data set into two separate regions and compute the mass in each region. The mass distribution at point x is estimated to be the sum of all 'weighted' masses from regions occupied by x, as a result of $n-1$ binary splits for a set of data of size n.

Let $x_1 < x_2 < \cdots < x_{n-1} < x_n$ on the real line[1], $x_i \in \mathcal{R}$ and $n > 1$. Let s_i be the binary split between x_i and x_{i+1}, yielding two non-empty regions having two masses m_i^L and m_i^R.

Definition 1. *Mass base function: $m_i(x)$ as a result of s_i, is defined as*

$$m_i(x) = \begin{cases} m_i^L & \text{if } x \text{ is on the left of } s_i \\ m_i^R & \text{if } x \text{ is on the right of } s_i \end{cases}$$

Note that $m_i^L = n - m_i^R = i$.

Definition 2. *Mass distribution: $mass(x_a)$ for a point $x_a \in \{x_1, x_2, \cdots, x_{n-1}, x_n\}$ is defined as a summation of a series of mass base function $m_i(x)$ weighted by $p(s_i)$ over $n-1$ splits as follows.*

$$\begin{aligned} mass(x_a) &= \sum_{i=1}^{n-1} m_i(x_a) p(s_i) \\ &= \sum_{i=a}^{n-1} m_i^L p(s_i) + \sum_{j=1}^{a-1} m_j^R p(s_j) \\ &= \sum_{i=a}^{n-1} i p(s_i) + \sum_{j=1}^{a-1} (n-j) p(s_j) \end{aligned} \quad (1)$$

$p(s_i)$ is the probability of selecting s_i. Note that we have defined $\sum_{i=q}^{r} f(i) = 0$, when $r < q$ for any function f.

Example. For an example of five points $x_1 < x_2 < x_3 < x_4 < x_5$, Figure 1 shows the resultant $m_i(x)$ due to each of the four binary splits s_1, s_2, s_3, s_4; and their associated masses over four splits are given below:

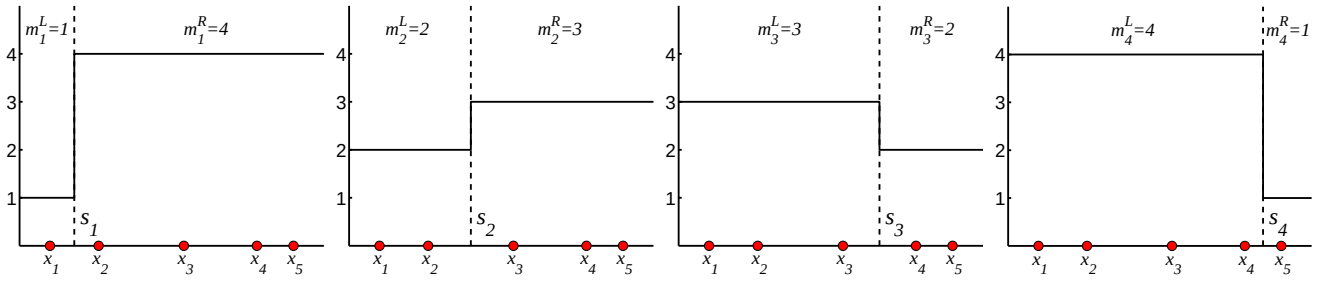
$$\begin{aligned} mass(x_1) &= 1p(s_1) + 2p(s_2) + 3p(s_3) + 4p(s_4) \\ mass(x_2) &= 4p(s_1) + 2p(s_2) + 3p(s_3) + 4p(s_4) \\ mass(x_3) &= 4p(s_1) + 3p(s_2) + 3p(s_3) + 4p(s_4) \\ mass(x_4) &= 4p(s_1) + 3p(s_2) + 2p(s_3) + 4p(s_4) \\ mass(x_5) &= 4p(s_1) + 3p(s_2) + 2p(s_3) + 1p(s_4) \end{aligned}$$

For a given data set, $p(s_i)$ can be estimated on the real line as $p(s_i) = (x_{i+1} - x_i)/(x_n - x_1) > 0$, as a result of random selection of splits based on a uniform distribution[2].

For a point $x \notin \{x_1, x_2, \cdots, x_{n-1}, x_n\}$, $mass(x)$ is defined as an interpolation between two masses of adjacent points x_i and x_{i+1}, where $x_i < x < x_{i+1}$.

[1] In data having a pocket of points of the same value, an arbitrary order can be 'forced' by adding multiples of an insignificant small value ϵ to each point of the pocket, without changing the general distribution.

[2] The estimated $mass(x)$ values can be calibrated to a finite data range Δ by multiplying a factor $(x_n - x_1)/\Delta$.



(a) $m_i(x)$ due to s_1 (b) $m_i(x)$ due to s_2 (c) $m_i(x)$ due to s_3 (d) $m_i(x)$ due to s_4

Figure 1: Examples of mass base function $m_i(x)$ due to each of the four binary splits: s_1, s_2, s_3, s_4.

Theorem 1. *$mass(x_a)$ is the maximum at $a = n/2$ for any density distribution of $\{x_1, \cdots, x_n\}$; and the points x_a, where $x_1 < x_2 < \cdots < x_{n-1} < x_n$ on the real line, can be ordered based on mass as follows.*

$$mass(x_a) < mass(x_{a+1}),\ a < n/2$$
$$mass(x_a) > mass(x_{a+1}),\ a > n/2$$

PROOF. The difference in mass between two subsequent points x_a and x_{a+1} differs in only one term, i.e., the mass for $p(s_a)$ only; and $\forall i \neq a$, the terms for $p(s_i)$ have the same mass.

$$\begin{aligned} & mass(x_a) - mass(x_{a+1}) \\ = & \quad \textstyle\sum_{i=a}^{n-1} ip(s_i) + \sum_{j=1}^{a-1}(n-j)p(s_j) \\ & - \textstyle\sum_{i=(a+1)}^{n-1} ip(s_i) - \sum_{j=1}^{a}(n-j)p(s_j) \\ = & \quad ap(s_a) - (n-a)p(s_a) \\ = & \quad (2a-n)p(s_a) \end{aligned} \tag{2}$$

Thus,

$$sign(mass(x_a) - mass(x_{a+1})) = \begin{cases} negative & \text{if } a < n/2 \\ 0 & \text{if } a = n/2 \\ positive & \text{if } a > n/2 \end{cases}$$

□

The point $x_{n/2}$ can be regarded as the median. Note that the number of points with the maximum mass depends on whether n is odd or even: When n is an odd integer, only one point has the maximum mass at x_{median}, where $median = \lceil n/2 \rceil$; when n is an even integer, two points have the maximum mass at $a = n/2$ and $a = 1 + n/2$.

Theorem 2. *$mass(x_a)$ is a concave function defined w.r.t. $\{x_1, x_2, \ldots, x_n\}$, when $p(s_i) = (x_{i+1} - x_i)/(x_n - x_1)$.*

PROOF. We only need to show that the gradient of $mass(x_a)$ is non-increasing, i.e., $g(x_a) > g(x_{a+1})$ for each a.

Let $g(x_a)$ the gradient between x_a and x_{a+1}, and from (2):

$$g(x_a) = \frac{mass(x_{a+1}) - mass(x_a)}{x_{a+1} - x_a} = \frac{n-2a}{x_n - x_1}$$

The result follows: $g(x_a) > g(x_{a+1})$ for $a \in \{1, 2, \ldots, n-1\}$.

□

Corollary 1. *A mass distribution estimated using binary splits stipulates an ordering, based on mass, of the points in a data cloud from $x_{n/2}$ (with the maximum mass) to the fringe points (with the minimum mass at either side of $x_{n/2}$), irrespective of the density distribution including uniform density distribution.*

Corollary 2. *The concavity of mass distribution stipulates that fringe points have markedly smaller mass than points close to $x_{n/2}$.*

The implication from Corollary 2 is that fringe points are 'stretched' to be farther away from the median in a mass space than in the original space—making it easier to separate fringe points from those points close to the median. (The mass space is mapped from the original space through $mass(x)$.) This property, otherwise hidden, can then be exploited by a data mining algorithm to achieve a better result for the intended task than the one without it. We will show that this simple mapping significantly improves the performance of four existing algorithms in information retrieval and regression tasks in Sections 4.1 and 4.2.

Equation (1) is sufficient to provide a mass distribution corresponds to a unimodal density function or a uniform density function. To better estimate multi-modal distributions, a high level mass distribution is required. This is provided in the following.

2.1.2 Level-h mass distribution estimation

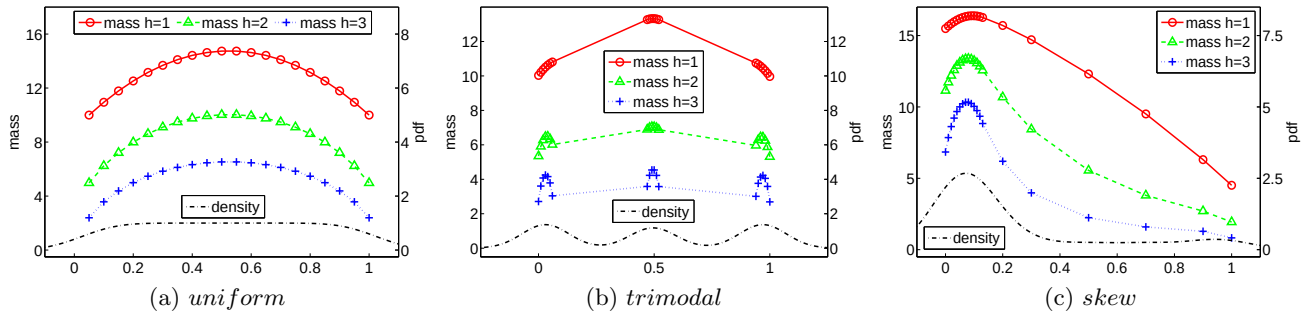
Definition 3. *Level-h mass distribution for a point $x_a \in \{x_1, \ldots, x_n\}$, where $h < n$, is expressed as*

$$\begin{aligned} mass(x_a, h) &= \sum_{i=1}^{n-1} mass_i(x_a, h\text{-}1)p(s_i) \\ &= \sum_{i=a}^{n-1} mass_i^L(x_a, h\text{-}1)p(s_i) + \\ &\quad \sum_{j=1}^{a-1} mass_j^R(x_a, h\text{-}1)p(s_j) \end{aligned} \tag{3}$$

Here a high level mass distribution is computed recursively by using the mass distributions obtained at lower levels. A binary split s_i in a level-h($>$1) mass distribution produces two level-(h-1) mass distributions: (a) $mass_i^L(x, h\text{-}1)$—the mass distribution on the left of split s_i which is defined using $\{x_1, \ldots, x_i\}$; and (b) $mass_i^R(x, h\text{-}1)$—the mass distribution on the right which is defined using $\{x_{i+1}, \ldots, x_n\}$. Equation (1) is the mass distribution at level-1.

Figure 2 shows part of the intermediate process in calculating $mass_i^L(x, h = 1)$ and $mass_i^R(x, h = 1)$ for two example splits $s_{i=7}$ and $s_{i=11}$ in order to obtain $mass(x, h = 2)$.

Using the same analysis in the proof for Theorem 1, the above equation can be re-expressed as:



(a) *uniform* (b) *trimodal* (c) *skew*

Figure 3: Examples of level-h mass distribution for $h = 1, 2, 3$ and density distribution from kernel density estimation: Gaussian kernel with bandwidth= 0.1. All three data sets have 20 points each.

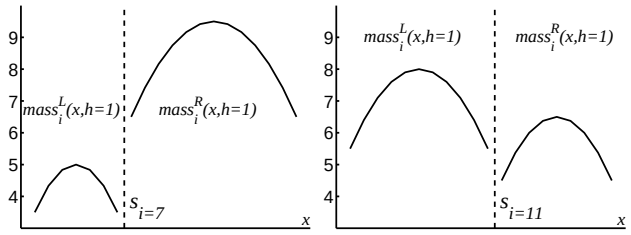


Figure 2: Two examples of $mass_i^L(x, h = 1)$ and $mass_i^R(x, h = 1)$ due to $s_{i=7}$ and $s_{i=11}$ in the process to get $mass(x, h = 2)$ from a data set of 20 points with uniform density distribution. The resultant $mass(x, h = 2)$ is shown in Figure 3(a).

$$mass(x_{a+1}, h) = mass(x_a, h) + \begin{cases} [mass_a^R(x_a, h\text{-}1) - mass_a^L(x_a, h\text{-}1)]p(s_a), & h > 1 \\ (n - 2a)p(s_a), & h = 1 \end{cases} \quad (4)$$

As a result, only the mass for the first point x_1 needs to be computed using Equation (3). Note that it is more efficient to compute the mass distribution from the above equation which has time complexity $O(n^{h+1})$; the computation using Equation (3) has $O(n^{h+2})$.

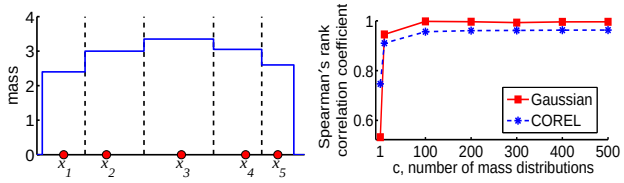
Definition 4. *A level-h mass distribution stipulates an ordering of the points in a data cloud from α-core points to the fringe points. The α-core point(s) of a data cloud have the highest mass value within α distance from the core point(s). A small α defines local core point(s); and a large α, which covers the entire value range for x, defines global core point(s).*

Examples of level-h mass estimation in comparison with kernel density estimation are provided in Figure 3. Note that $h = 1$ mass estimation treats the entire data as a group, and it produces a concave function. As a result, an $h = 1$ mass estimation always has its global core point(s) at the median, regardless of the underlying density distribution—see three examples of $h = 1$ mass estimation in Figure 3.

For $h > 1$ mass distribution, though there is no guarantee for a single concave function for the entire data set, each cluster within the data cloud still exhibits a concave function and it becomes more distinct (as a concave function) as h increases. This is shown in Figure 3(b) which has a trimodal density distribution. Notice that the $h > 1$ mass distributions have three α-core points for some α, e.g., 0.2.

Traditionally, one can determine the core-ness or the fringe-ness of a non-uniformly distributed data to some degree by using density or distance (but not in uniform density distribution.) Mass allows one to do that in any distributions without density or distance calculation—the key computational expense in all methods that employ them. For example in Figure 3(c) which has a skew density distribution, the distinction between near fringe points and far fringe points are less obvious using density, unless distances are computed to reveal the difference. In contrast, mass distribution depicts the relative distance from x_{median} using the fringe points' mass values, without further calculation.

This section has described properties of mass distribution from a theoretical perspective. Though it is possible to estimate mass distribution using Equations (1) and (3), they are limited by its high computational cost. We suggest a practical mass estimation method in the next section. We use the term 'mass estimation' and 'mass distribution estimation' interchangeably hereafter.



(a) (b)

Figure 4: (a) An example of practical mass distribution $mass(x, h|\mathcal{D})$ for 5 points, assuming a rectangular function for each point. (b) Correlation between the orderings provided by $mass(x, 1)$ and $\overline{mass}(x, 1)$ for two data sets: one-dimensional Gaussian density distribution and the COREL data set used in Section 4.1 (whose result is averaged over 67 dimensions).

2.2 Practical mass estimation

Here we devise an approximation to Equation (3) using random subsamples from the given data set.

Definition 5. *$mass(x, h|\mathcal{D})$ is the approximate mass distribution for a point $x \in \mathcal{R}$, defined w.r.t. $\mathcal{D} = \{x_1, \ldots, x_\psi\}$, where $\mathcal{D}$ is a random subset of the given data set D, and $\psi \ll |D|$, $h < \psi$.*

Assume a rectangular function for each point $x \in \mathcal{D}$ (as shown in Figure 4(a)), $mass(x, h|\mathcal{D})$ is implemented using a lookup table with each rectangle function covers a range $(x_{i-1} + x_i)/2 \leq x < (x_{i+1} + x_i)/2$ for each point $x_i \in \mathcal{D}$ having the same $mass(x_i, h|\mathcal{D})$ value. The range for each of the two end-points is set to have equal length on either side of the point. In addition, a number of mass distributions

needs to be constructed from different samples in order to have a good approximation, that is,

$$\overline{mass}(x,h) = \frac{1}{c}\sum_{k=1}^{c} mass(x,h|\mathcal{D}_k) \quad (5)$$

The computation of $mass(x,h)$ using the given data set D costs $O(|D|^{h+1})$; whereas $\overline{mass}(x,h)$ costs $O(c\psi^{h+1})$.

Only relative, not absolute, mass is required to provide an ordering between instances. Because the relative mass is w.r.t. median and median is a robust estimator [1]—that is why small subsamples produce a good order estimator.

Figure 4(b) shows the correlation (in terms of Spearman's rank correlation coefficient) between the orderings provided by $mass(x,1)$ using the entire data set and $\overline{mass}(x,1)$ using $\psi = 8$ in two data sets, each having 10000 data points. They achieve very high correlations when $c \geq 100$.

The ability to use a small sample, rather than a large sample, is a key characteristic of mass estimation. We show in this paper that $mass(x,h|\mathcal{D})$ can be employed very effectively for three different tasks: information retrieval, regression and anomaly detection, through a mass-based formalism to be described in the next section. Although the mass estimation is designed for one dimension only, we show that it can be employed to solve multi-dimensional problems.

3. MASS-BASED FORMALISM

Let $\mathbf{x}_i = [x_i^1, \ldots, x_i^u]$; $\mathbf{x}_i \in D$; and $\mathbf{z}_i = [z_i^1, \ldots, z_i^t]$; $\mathbf{z}_i \in D'$. The proposed formalism consists of three components:

C1 The first component constructs a number of mass distributions. A mass distribution $mass(x^d, h|\mathcal{D})$ for dimension d is obtained using our proposed mass estimation, as given in Definition 5. A total number of t mass distributions is generated which forms $\widetilde{\mathbf{mass}}(\mathbf{x}) \rightarrow \mathcal{R}^t$, where $t \gg u$. This procedure is given in Algorithm 1.

C2 The second component maps the data set D in the original space of u dimensions into a new data set D' of t dimensions using $\widetilde{\mathbf{mass}}(\mathbf{x}) = \mathbf{z}$. This procedure is described in Algorithm 2.

C3 The third component employs a decision rule to determine the final outcome for the task at hand. It is a task-specific decision function applied to $\mathbf{z}$ in the new feature space.

Algorithm 1 : Mass_Estimation(D, ψ, h, t)

Inputs: D - input data; ψ - data size for $\mathcal{D}_k$; h - level of mass distribution; t - number of mass distributions.
Output: $\widetilde{\mathbf{mass}}(\mathbf{x}) \rightarrow \mathcal{R}^t$ - a function consists of t mass distributions, $mass(x^d, h|\mathcal{D}_k)$.

1: **for** $k = 1$ to t **do**
2: $\quad \mathcal{D}_k \leftarrow$ a random subset of size ψ from D;
3: $\quad d \leftarrow$ a randomly selected dimension from { 1,. . . ,u };
4: $\quad$ Build $mass(x^d, h|\mathcal{D}_k)$;
5: **end for**

The formalism becomes a blueprint for different tasks. Components **C1** and **C3** are mandatory in the formalism, but component **C2** is optional, depending on the task.

For information retrieval and regression, the task-specific **C3** procedure is simply using an existing algorithm for the task except that the process is carried out in the new mapped mass space, instead of the original space. This procedure is given in Algorithm 3. The task-specific **C3** procedure for anomaly detection is shown in steps 2-5 in Algorithm 4. Note that anomaly detection requires **C1** and **C3** only; whereas the other two tasks require all three components.

Algorithm 2 : Mass_Mapping$(D, \widetilde{\mathbf{mass}})$

Inputs: D - input data; $\widetilde{\mathbf{mass}}$ - a function consists of t mass distributions, $mass(x^d, h|\mathcal{D})$.
Output: D' - a set of mapped instances $\mathbf{z}_i$ in t dimensions.

1: **for** $i = 1$ to $|D|$ **do**
2: $\quad \mathbf{z}_i \leftarrow \widetilde{\mathbf{mass}}(\mathbf{x}_i)$;
3: **end for**

Algorithm 3 : Perform task in MassSpace(D, ψ, h, t)

Inputs: D - input data; ψ - data size for $\mathcal{D}$; h - level of mass distribution; t - number of mass distributions.
Output: Task-specific model.

1: $\widetilde{\mathbf{mass}}(.) \leftarrow$ Mass_Estimation(D, ψ, h, t);
2: $D' \leftarrow$ Mass_Mapping$(D, \widetilde{\mathbf{mass}})$;
3: Perform task (information retrieval or regression) in the mapped mass space using D';

Algorithm 4 for Anomaly Detection : MassAD(D, ψ, h, t)

Inputs: D - input data; ψ - data size for $\mathcal{D}$; h - level of mass distribution; t - number of mass distributions.
Output: Ranked instances in D.

1: $\widetilde{\mathbf{mass}}(.) \leftarrow$ Mass_Estimation(D, ψ, h, t);
2: **for** $i = 1$ to $|D|$ **do**
3: $\quad \mathsf{m}_i \leftarrow$ Average of t masses from $\widetilde{\mathbf{mass}}(\mathbf{x}_i)$;
4: **end for**
5: Rank instances in D based on m_i with low mass denotes anomalies and high mass denotes normal points;

4. EXPERIMENTS

We evaluate the performance of MassSpace and MassAD for three tasks in the following three subsections. In information retrieval and regression tasks, the mass estimation uses $\psi = 8$ and $t = 1000$. These settings are obtained by examining the rank correlation as shown in Figure 4(b)—having a high rank correlation between $mass(x,1)$ and $\overline{mass}(x,1)$. Note that this is done before any method is applied and no further fine-tuning. In anomaly detection tasks, $\psi = 256$ and $t = 100$ are used so that they are comparable to those used in a benchmark method for a fair comparison. $h = 1$ is used in all tasks, unless stated otherwise. All the experiments are run in Matlab and conducted on a Pentium 4 machine with an AMD Opteron machine with a 1.8 GHz processor and 4 GB memory. The performance of each method is measured in terms of task-specific performance measure and runtime. Paired t-tests at 5% significance level are conducted to examine whether the difference in performance is significant between two algorithms under comparison.

Note that we treat information retrieval and anomaly detection as unsupervised learning tasks. Classes/labels in the original data are used as ground truth for evaluation of performance only; they are not used in building mass distributions. In regression, only the training set is used to build mass distributions in step 1 of Algorithm 3; the mapping in step 2 is conducted for both the training and testing sets.

Table 2: CBIR results (the higher the better for BEP.)

	BEP ($\times 10^{-2}$)						Processing time (second)					
	MRBIR′	MRBIR	Qsim′	Qsim	InstR′	InstR	MRBIR′	MRBIR	Qsim′	Qsim	InstR′	InstR
One query	**11.52**	9.69	**10.31**	7.78	**10.31**	7.78	1.980	**1.111**	0.410	**0.034**	0.410	**0.034**
Round 1	**15.14**	12.72	**15.39**	10.59	**13.45**	9.40	2.499	**2.155**	0.588	**0.078**	0.558	**0.046**
Round 2	**16.81**	13.90	**17.46**	11.81	**15.07**	9.99	2.501	**2.155**	0.646	**0.139**	0.559	**0.047**
Round 3	**17.94**	14.75	**18.46**	12.59	**16.15**	10.36	2.499	**2.155**	0.737	**0.227**	0.560	**0.048**
Round 4	**18.74**	15.33	**19.18**	13.16	**16.96**	10.78	2.501	**2.155**	0.862	**0.355**	0.561	**0.049**
Round 5	**19.39**	15.71	**19.62**	13.55	**17.62**	11.05	2.499	**2.155**	1.016	**0.516**	0.562	**0.050**

Table 3: Regression results (the smaller the better for MSE; the larger the better for SCC.)

	data		MSE ($\times 10^{-2}$)			SCC ($\times 10^{-2}$)			Processing time		Factor increase	
	size	u	SVR′	SVR	W/D/L	SVR′	SVR	W/D/L	SVR′	SVR	time	dimension
tic	9822	85	**5.58**	5.62	17/0/3	**2.12**	1.07	18/0/2	63.61	**29.85**	2.1	12
wine_white	4898	11	**1.21**	1.36	20/0/0	**45.18**	38.60	20/0/0	17.30	**7.24**	2.4	91
quake	2178	3	**2.86**	2.92	18/0/2	**0.84**	0.31	14/0/6	3.18	**1.09**	2.9	333
wine_red	1599	11	1.62	1.62	11/0/9	**38.20**	37.76	13/0/7	2.00	**0.76**	2.6	91
concrete	1030	8	**0.33**	0.57	20/0/0	**92.62**	87.17	20/0/0	1.08	**0.44**	2.5	125

4.1 Content-Based Image Retrieval

We use a Content-Based Image Retrieval (CBIR) task as an example of information retrieval. The MassSpace approach is compared with three state-of-the-art CBIR methods that deal with relevance feedbacks: a manifold based method MRBIR [9], and two recent techniques for improving similarity calculation, i.e., Qsim [19] and InstRank [8]; and we employ the Euclidean distance to measure the similarity between instances in these two methods. The default parameter settings are used for all these methods. Because the same CBIR method is employed in the mapped space in the MassSpace approach, we denote them as MRBIR′, Qsim′ and InstRank′ for those employ MRBIR, Qsim and InstRank, respectively.

Our experiments are conducted using the COREL image database [18] of 10000 images, which contains 100 categories and each category has 100 images. Each image is represented by a 67-dimensional feature vector, which consists of 11 shape, 24 texture and 32 color features. To test the performance, we randomly select 5 images from each category to serve as the initial queries. For a query, the images within the same category are regarded as relevant and the rest are irrelevant. For each query, we continue to perform 5 rounds of relevance feedback. In each round, 2 positive and 2 negative feedbacks are provided. This relevance feedback process is also repeated 5 times, each up to 5 feedback rounds. Finally, the average results with one query and in different feedback rounds are recorded. The retrieval performance is measured in terms of Break-Even-Point (BEP) [19, 18] of the precision-recall curve. The online processing time reported is the time required in each method for a query plus the stated feedback rounds. The reported result is an average over 5×100 runs for query only; and an average over $5 \times 100 \times 5$ runs for query plus feedbacks. The offline costs of constructing the mass distributions and the mapping of 10000 images are 2.87 and 1.25 seconds, respectively.

The results are presented in Table 2 where the best performance at each round has been boldfaced. The results are grouped in pairs for ease of comparison.

The BEP results clearly show that the MassSpace approach achieves a better retrieval performance than that using the original space in all three methods MRBIR, Qsim and InstR, regardless it is with one query only or in relevance feedbacks. Paired t-tests at 5% significance level also indicate that the MassSpace approach significantly outperforms each of the three methods in all experiments, without exception. These results show that the mass space provides useful additional information that is hidden in the original space.

The processing time for each of the three methods in the mass space is expected to be longer than that in the original space because the number of dimensions in the mass space is significantly higher than those in the original space, where $t = 1000$ and $u = 67$.

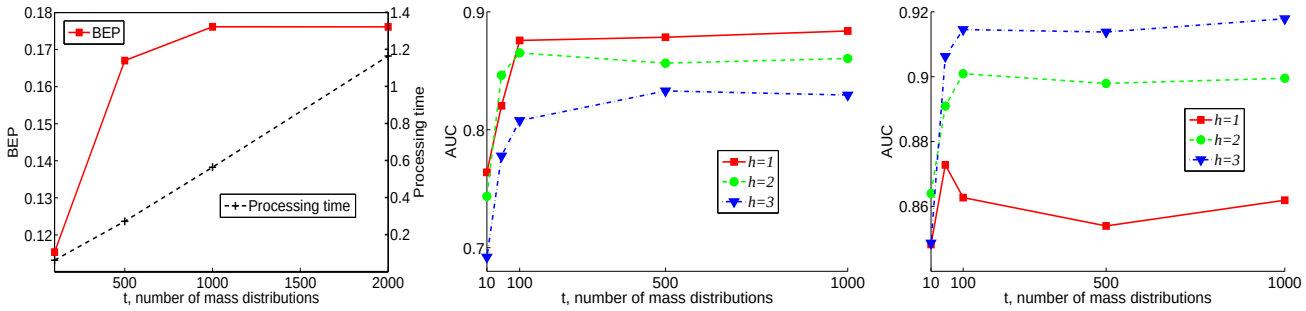
Figure 5(a) shows an example of performance for InstR′—BEP increases as t increases until it reaches a plateau at some t value; and the processing time for InstR′ is linear w.r.t. the number of dimensions of the mass space, t.

4.2 Regression

In this experiment, we compare SVR′ with SVR—support vector regression [16] that employs the mapped mass space versus that employs the original space. SVR is the ϵ-SVR algorithm with RBF kernel, implemented by LIBSVM [6]. SVR is chosen here because it is one of the top performing regression models.

We utilize five benchmark data sets including four selected from UCI repository [2] and one earthquake data [14] from www.cs.waikato.ac.nz/ml/weka/ distribution. The data characteristics are summarized in the first three columns of Table 3. We select only those data sets which are more than 1000 data points with all real-valued attributes and without missing values—in order to get a result with a higher confidence than those obtained from small data sets.

On each data set, we randomly sample two-thirds of the instances for training and the remaining one-third for testing. This is repeated 20 times and we report the average result of these 20 runs. The data set, whether in the original space or the mass space, is min-max normalized before an ϵ-SVR model is trained. To select optimal parameters for the ϵ-SVR algorithm, we conduct a 5-fold cross validation based on mean squared error using the training set only. The kernel parameter γ is searched in the range $\{2^{-15}, 2^{-13}, 2^{-11}, \cdots, 2^3, 2^5\}$; the regularization parameter



(a) An example of CBIR-round 5 result. (b) Effect of h in the Forest data set. (c) Effect of h in the Smtp data set.

Figure 5: (a) The retrieval performance and the processing time as t increases for `InstR'`. (b) High h produces a poorer detection performance in this case. (c) High h produces a better detection performance in this case.

Table 4: Data characteristics of the data sets in anomaly detection tasks. The percentage in brackets indicates the percentage of anomalies.

	data size	u	anomaly class
Http	567497	3	attack (0.4%)
Forest	286048	10	class 4 (0.9%) vs class 2
Mulcross	262144	4	2 clusters (10%)
Smtp	95156	3	attack (0.03%)
Shuttle	49097	9	classes 2,3,5,6,7 (7%) vs class 1

C in the range $\{0.1, 1, 10\}$, and ϵ in the range $\{0.01, 0.05, 0.1\}$. We measure regression performance in terms of mean squared error (MSE) and squared correlation coefficient (SCC), and runtime in seconds. The runtime reported is the runtime for `SVR` only. The total cost of mass estimation (from the training set) and mapping (of training and testing sets) is 3.95 seconds in the largest data set, tic. The cost of normalisation and the parameter search using 5-fold cross-validation is not included in the reported result for both `SVR'` and `SVR`.

The result is presented in Table 3. `SVR'` performs significantly better than `SVR` in all data sets in both MSE and SCC measures; the only exception is in the wine_red data set. Although `SVR'` takes more time to run as it runs on the data with a significantly higher dimension, yet the factor of increase in time ranges from 2 to 3 only when the factor of increase in the number of dimensions ranges from 12 to over 300 (shown in the last two columns of Table 3). This is because the time complexity in the key optimisation process in `SVR` is not dependent on the number of dimensions.

4.3 Anomaly Detection

This experiment compares `MassAD` with four state-of-the-art anomaly detectors: isolation forest (or `iForest`) [10], a distance-based method `ORCA` [3], a density-based method `LOF` [4], and one-class support vector machine (or `1-SVM`) [13]. `MassAD` is built with $t = 100$ and $\psi = 256$, the same default settings as used in `iForest` [10], which also employs a multi-model approach. The parameter settings employed for `ORCA` and `LOF` are as stated in [10]. `1-SVM` uses Radial Basis Function kernel and an inverse width parameter estimated by the method suggested in [5].

All the methods are tested on the five largest data sets used in [10]. The data characteristics are summarized in Table 4, which include one anomaly data generator Mulcross [12] and the other four are from UCI repository [2]. The performance is evaluated in terms of averaged AUC (area under ROC curve) and processing time (a total of training time and testing time) over ten runs (following [10]). `MassAD` and `iForest` are implemented in Matlab and tested on an AMD Opteron machine with a 1.8 GHz processor and 4 GB memory. The results for `ORCA`, `LOF` and `1-SVM` are conducted using the same experimental setting but on a faster 2.3 GHz machine, the same machine used in [10].

Table 5: AUC values for anomaly detection.

	`MassAD`		`iForest`	`ORCA`	`LOF`	`1-SVM`
	ψ=8	ψ=256				
Http	0.98	**1.00**	**1.00**	0.36	N/A	0.90
Forest	0.88	**0.91**	0.87	0.83	0.57	0.90
Mulcross	0.97	**0.99**	0.96	0.33	0.59	0.59
Smtp	0.86	0.86	**0.88**	0.87	0.32	0.78
Shuttle	0.99	0.99	**1.00**	0.60	0.55	0.79

Table 6: Runtime (second) for anomaly detection.

	`MassAD`		`iForest`	`ORCA`	`LOF`	`1-SVM`
	ψ=8	ψ=256				
Http	**27**	34	147	9487	> 2weeks	35872
Forest	**14**	18	79	6995	224380	9738
Mulcross	**13**	17	75	2512	156044	7343
Smtp	**4**	7	26	267	24281	987
Shuttle	**2**	4	15	157	7490	333

The AUC values of all methods are presented in Table 5 where the figures boldfaced are the best performance for each data set. The results show that `MassAD` with $\psi = 256$ achieves the best performance on the three largest data sets; and even on the other two data sets, `MassAD` is also competitive since the AUC gap is small between `MassAD` and the best method, i.e., `iForest`. It is noteworthy that `MassAD` significantly outperforms the traditional density-based, distance-based and SVM anomaly detectors in all data sets, except two: one in Smtp when compared with `ORCA` and another in Forest when compared with `1-SVM`. The above observations validate the effectiveness of our proposed mass estimation on anomaly detection tasks.

Table 6 shows the runtime result. Although `MassAD` is run on a slower machine, it still has a significant advantage in term of processing time over `ORCA`, `LOF` and `1-SVM`. The comparison with `iForest` is presented in Table 7 with a breakdown of training time and testing time. Note that `MassAD` takes the same time as `iForest` in training, but it only takes about one-tenth of the time required by `iForest` in testing. These results show that `MassAD` is an efficient anomaly detector.

Figures 5(b) and 5(c) show the effect of h on the detection

Table 7: Training time and testing time (second) for `MassAD` and `iForest`, using $t = 100$ and $\psi = 256$.

	Training time		Testing time	
	`MassAD`	`iForest`	`MassAD`	`iForest`
Http	20.96	19.72	12.93	127.47
Forest	11.26	11.47	6.97	67.45
Mulcross	10.54	10.69	6.82	64.34
Smtp	4.97	4.1	2.22	22.39
Shuttle	3.43	3.23	1.01	11.79

performance of `MassAD` with $\psi = 8$—higher h degrades the detection performance in Forest; but it improves in Smtp. This shows that for best performance in individual data set, some parameter tuning is required, like most other algorithms. Note that there is no attempt to tune this parameter (or any other parameters) in the result reported in Tables 5, 6 and 7 where $h = 1$ is used throughout.

The time and space complexities for four methods are given in Table 8. `MassAD` and `iForest` have the best time and space complexities due to their ability to use small $\psi \ll n$ and $h = 1$. Note that `MassAD` ($h = 1$) is faster by a factor of $log(\psi = 256) = 8$ which shows up in the testing time—ten times faster than `iForest` given in Table 7. The training time disadvantage, compared to iForest, did not show up because of small ψ. `MassAD` also has an advantage over `iForest` in space complexity by a factor of $log(\psi)$.

Table 8: A comparison of time and space complexities. The time complexity includes both training and testing. n is the given data set size and u is the number of dimensions. For `MassAD` and `iForest`, the first part of the summation is the training time and the second the testing time.

	Time complexity	Space complexity
`MassAD`	$O(t(\psi^{h+1} + n))$	$O(t\psi)$
`iForest`	$O(t(\psi + n) \cdot log(\psi))$	$O(t\psi \cdot log(\psi))$
`ORCA`	$O(un \cdot log(n))$	$O(un)$
`LOF`	$O(un^2)$	$O(un)$

4.4 Constant time and space complexities

In this section, we show that $mass(x, h|\mathcal{D})$ (in step 4 of Algorithm 1) takes only constant time, regardless of the given data size n, when the algorithmic parameters are fixed. Table 9 reports the runtime time for sampling (to get a random sample of size ψ from the given data set—steps 2 and 3 of Algorithm 1) and the runtime for mass estimation—to construct $mass(x, h|\mathcal{D})$ t times, for five data sets which include the largest and smallest data sets in regression and anomaly detection tasks.

The results show that the sampling time increases linearly with the size of the given data set, and it takes a significantly longer (in the largest data set) than the time to construct the mass distribution—which is constant, regardless of the given data size. Note that the training time provided in Table 7 includes both the sampling time and mass estimation time, and it is dominated by the sampling time.

The memory required for each construction of $mass(x, h|\mathcal{D})$ is to store one lookup table of size ψ which is constant, again independent of the given data size.

Table 9: Runtime (second) for sampling, $mass(x, 1|\mathcal{D})$ and $mass(x, 3|\mathcal{D})$, where $t = 1000$ and $\psi = 8$.

	data size	sampling	$mass(x, 1\|\mathcal{D})$	$mass(x, 3\|\mathcal{D})$
Http	567497	185.21	0.57	17.15
Shuttle	49097	12.47	0.59	17.37
COREL	10000	2.34	0.53	17.28
tic	9822	2.28	0.56	17.23
concrete	1030	0.36	0.48	17.28

Summary

The above results in all three tasks show that the orderings provided by mass distributions deliver additional information about the data that would otherwise hidden in the original features. The additional information improves the task-specific performance significantly, especially in the information retrieval and regression tasks.

Using Algorithm 3, the runtime is expected to be higher because the new space has much higher dimensions than the original space ($t \gg u$). It shall be noted that the runtime increase (linearly or worse) is solely a characteristic of the existing algorithms used, not due to the mass space mapping which has constant time and space complexities.

We believe that a more tailored approach that better integrates the information provided by mass (into the **C3** component in the formalism) for the specific task can potentially further improve the current level of performance in terms of either task-specific performance measure or runtime. We have demonstrated this 'direct' application using Algorithm 4 for the anomaly detection task, in which `MassAD` performs equally well or significantly better than four state-of-the-art methods in terms of task-specific performance measure, and it executes faster than all other methods in terms of runtime.

Why does one-dimensional mapping work when tackling multi-dimensional problems? The mapping transforms each original feature to approximately $\frac{t}{u}$ features in the mass space—unearth hidden information for each original feature. It is more of a question whether an algorithm can make full use of this information in the new space; as both the original and new spaces are multi-dimensional. A multi-dimensional mapping may better enhance information in some domains. It is thus worthwhile to explore this extension.

5. RELATION TO KERNEL DENSITY ESTIMATION

A comparison of mass estimation and kernel density estimation is provided in Table 10.

Table 10: A comparison of kernel density estimation and mass estimation. Kernel density estimation requires two parameter settings: kernel function $K(.)$ and bandwidth h_w; mass estimation has one: h.

Kernel density$(x) = \frac{1}{nh_w} \sum_{i=1}^{n} K(\frac{x-x_i}{h_w})$
$mass(x, h) = \begin{cases} \sum_{i=1}^{n-1} mass_i(x, h\text{-}1)p(s_i), & h > 1 \\ \sum_{i=1}^{n-1} m_i(x)p(s_i), & h = 1 \end{cases}$

Like kernel estimation, mass estimation at each point is computed through a summation of a series of values from a mass base function $m_i(.)$, equivalent to a kernel function $K(.)$. The two methods differ in the following ways:

Table 11: CBIR results: Compare with Qsim″ and InstR″ which use Gaussian kernel density estimation.

	BEP ($\times 10^{-2}$)						Processing time (second)					
	Qsim′	Qsim″	Qsim	InstR′	InstR″	InstR	Qsim′	Qsim″	Qsim	InstR′	InstR″	InstR
One Query	**10.31**	2.51	7.78	**10.31**	2.51	7.78	0.410	0.409	**0.034**	0.410	0.409	**0.034**
Round 1	**15.39**	2.72	10.59	**13.45**	2.66	9.40	0.588	0.633	**0.078**	0.558	0.571	**0.046**
Round 2	**17.46**	2.67	11.81	**15.07**	2.51	9.99	0.646	0.780	**0.139**	0.559	0.574	**0.047**
Round 3	**18.46**	2.56	12.59	**16.15**	2.31	10.36	0.737	0.989	**0.227**	0.560	0.577	**0.048**
Round 4	**19.18**	2.53	13.16	**16.96**	2.20	10.78	0.862	1.275	**0.355**	0.561	0.580	**0.049**
Round 5	**19.62**	2.46	13.55	**17.62**	2.07	11.05	1.016	1.629	**0.516**	0.562	0.582	**0.050**

Table 12: Anomaly detection: MassAD vs DensityAD.

	AUC		Time (second)	
	MassAD	DensityAD	MassAD	DensityAD
Http	1.00	0.99	34	33
Forest	**0.91**	0.69	18	18
Mulcross	0.99	1.00	17	17
Smtp	**0.86**	0.60	7	7
Shuttle	**0.99**	0.92	4	4

- *Aim*: Kernel estimation is aimed to do probability density estimation; whereas mass estimation is to estimate an order from the core points to the fringe points.

- *Kernel function*: While kernel estimation can use different kernel functions for probability density estimation; we doubt that mass estimation requires a different base function for two reasons. First, a more sophisticated function is unlikely to provide a better ordering than a simple rectangular function. Second, the rectangular function keeps the computation simple and fast. In addition, a kernel function must be fixed (i.e., having user-defined values for its parameters); e.g., the rectangular kernel function has fixed width or fixed per unit size. But the rectangular function used in mass has no parameter and no fixed width.

- *Sample size*: Kernel estimation or other density estimation methods require a large sample size in order to estimate the probability accurately [7]. Mass estimation using $mass(x, h|\mathcal{D})$ needs only a small sample size in an ensemble to accurately estimate the ordering.

- *Definition*: Probability density can be defined independent of data, whereas mass (in its current form) must be defined w.r.t. a set of data.

Because of a lack of concavity, density will not perform as successfully as mass. Here we present the results using a Gaussian kernel density estimation, replacing $mass(x, h|\mathcal{D}_k)$, using the same subsample size in an ensemble approach. The bandwidth parameter is set to be the standard deviation of the subsample; and all the other parameters are the same.

The results for information retrieval and anomaly detection are provided in Tables 11 and 12. Compare to mass, density performs significantly worse in information retrieval task in all experiments using Qsim and InstR, denoted as Qsim″ and InstR″, respectively. They are even worse than those run in the original space. In anomaly detection, DensityAD, which uses a Gaussian kernel density estimation, performs significantly worse than MassAD in three out of five data sets in the anomaly detection tasks, and equally well in the other two data sets.

6. RELATED WORK

There is a close relationship between the proposed mass and data depth [11]: they both delineate the centrality of a data cloud (as opposed to compactness in the case of density.) The properties common to both measures are: (a) the centre of a data cloud has the maximum value of the measure; (b) an ordering from the centre (having the maximum value) to the fringe points (having the minimum values).

However, there are three fundamental differences. First, data depth can deal with unimodal data only; whereas mass can deal with both unimodal and multi-modal data by setting $h = 1$ or $h > 1$.

Second, mass is a simple and straightforward measure, and has an efficient estimation method; whereas data depth has many different definitions, depending on the construct used to define depth. The constructs could be Mahalanobis, Convex Hull, simplicial and so on [11], all of which are expensive to compute [1]—this has been the main obstacle in applying data depth for real applications in multi-dimensional problems. In addition, the centre of a data cloud varies depending on the construct used to define data depth; whereas mass ($h = 1$) always has the centre located at the mid-point in the series of data points.

Third, the $h = 1$ mass estimation guarantees concavity—the reason why a simple mass space mapping improves the task-specific performance of four existing algorithms in information retrieval and regression tasks. In contrast, there is no such guarantee in data depth. Because of a lack of concavity, like density, data depth is unlikely to be as successful as mass in the three tasks we have reported here, even if we ignore the runtime issue.

Mass estimation can be implemented in different ways. For example, we have reported an implementation using a tree structure (instead of a lookup table) in [15] using Half-Space Trees. It reduces the time complexity to $O(th(\psi+n))$ from $O(t(\psi^{h+1}+n))$, making it feasible for very high level-h mass estimation. We have repeated the experiments reported in this paper using Half-Space Trees, and it produces almost identical results.

Half-Space Trees extends naturally from one-dimensional mass estimation to multi-dimensional mass estimation. This has been tested in anomaly detection task [15].

iForest [10] and MassAD shares some common features: Both are ensemble methods which build t models, each from a random sample of size ψ, and they both combine the outputs of the models through averaging during testing. Although iForest [10] is designed specifically for anomaly detection which employs path length—an instance traverses from the root of a tree to its leaf—as the anomaly score, we have shown in [15] that the path length used in iForest is in fact a proxy to mass. In other words, iForest is a kind of

mass-based method—that is why `MassAD` and `iForest` have similar detection accuracy.

We have already established a direct application of mass in content-based image retrieval [17]. In addition to the mass-space mapping we have shown here, [17] presents a framework that assigns a weight (based on `iForest`, thus, mass) to each feature w.r.t. a query image; and then it ranks images in the database according to their weighted average feature values. The framework also incorporates relevance feedback which modifies the ranking based on the feedbacks through reweighted features. This framework makes use of all three components of the formalism stated in Section 3. This direct application of mass performs significantly better than the indirect approach we have shown in Section 4.1, in terms of both retrieval performance and processing time. Like `MassAD`, no distance calculations are used at all—the key reason for its superior time complexity.

7. CONCLUSIONS AND FUTURE WORK

This paper makes two key contributions. First, we introduce a base measure, mass, and delineate its three properties: (i) a mass distribution stipulates an ordering from core points to fringe points in a data cloud; (ii) this ordering accentuates the fringe points with a concave function—the essential property that is easily exploited by existing algorithms to improve their task-specific performance; and (iii) it is a constant-time-and-space-complexities estimation method. Density estimation has been the base modelling mechanism employed in many techniques thus far. Mass estimation introduced here provides an alternative choice, and it is better suited for many tasks which require an ordering rather than probability density estimation.

Second, we present a mass-based formalism which forms a basis to apply mass for different tasks. The three tasks (i.e., information retrieval, regression and anomaly detection) in which we have successfully applied are just examples of its application. Mass estimation has potentials in applications as diverse as density estimation has applied now.

There are potential extensions to the current work. First, one shall consider a new way to best utilise mass when solving a problem. In other words, we advocate a direct application of mass, rather than an indirect application. Second, the algorithms provided here for the three tasks are by no means definitive, and even the formalism can be improved or extended to include more tasks. Third, because the purposes and their properties differ, mass estimation is not intended to replace density estimation—it is thus important to identify areas in which each is best suited for. This will ascertain areas in which density has been a mismatch, unbeknown thus far.

Acknowledgements

This work is partially supported by the Air Force Research Laboratory, under agreement# FA2386-10-1-4052. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Phil Rayment, David Albrecht, Zhouyu Fu and Geoff Webb have provided many helpful comments in the early draft. Suggestions from the anonymous reviewers have helped to improve the clarity of this paper.

The Matlab source code of mass estimation is available at http://sourceforge.net/projects/mass-estimation/.

8. REFERENCES

[1] G. Aloupis. Geometric measures of data depth. *DIMACS Series in Discrete Math and Theoretical Computer Science*, 72:147–158, 2006.

[2] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[3] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of SIGKDD*, pages 29–38, 2003.

[4] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying density-based local outliers. In *Proceedings of SIGKDD*, pages 93–104, 2000.

[5] B. Caputo, K. Sim, F. Furesjo, and A. Smola. Appearance-based object recognition using svms: which kernel should i use? In *NIPS workshop on Statitsical methods for computational experiments in visual processing and computer vision*, 2002.

[6] C.-C. Chang and C.-J. Lin. LIBSVM: a library for support vector machines, 2001.

[7] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Second Edition. John Wiley, 2001.

[8] G. Giacinto and F. Roli. Instance-based relevance feedback for image retrieval. In *Advances in NIPS*, pages 489–496, 2005.

[9] J. He, M. Li, H. Zhang, H. Tong, and C. Zhang. Manifold-ranking based image retrieval. In *Proceedings of ACM Multimedia*, pages 9–16, 2004.

[10] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *Proceedings of ICDM*, pages 413–422, 2008.

[11] R. Liu, J. M. Parelius, and K. Singh. Multivariate analysis by data depth. *The Annals of Statistics*, 27(3):783–840, 1999.

[12] D. M. Rocke and D. L. Woodruff. Identification of outliers in multivariate data. *Journal of the American Statistical Association*, 91(435):1047–1061, 1996.

[13] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt. Support vector method for novelty detection. In *Advances in NIPS*, pages 582–588, 2000.

[14] J. S. Simonoff. *Smoothing Methods in Statistics*. Springer-Verlag, 1996.

[15] K. M. Ting, S. C. Tan, and F. T. Liu. Mass: A new ranking measure for anomaly detection. *Gippsland School of Information Technology, Monash University*, Technical Report TR2009/1, 2009.

[16] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Second Edition. Springer, 2000.

[17] G.-T. Zhou, K. M. Ting, F. T. Liu, and Y. Yin. Relevance feature mapping for content-based image retrieval. In *Proceedings of Multimedia Data Mining Workshop at KDD*, 2010.

[18] Z.-H. Zhou, K.-J. Chen, and H.-B. Dai. Enhancing relevance feedback in image retrieval using unlabeled data. *ACM Transactions on Information Systems*, 24(2):219–244, 2006.

[19] Z.-H. Zhou and H.-B. Dai. Query-sensitive similarity measure for content-based image retrieval. In *Proceedings of ICDM*, pages 1211–1215, 2006.

Multi-Dimensional Mass Estimation and Mass-based Clustering

Kai Ming Ting and Jonathan R. Wells
Gippsland School of Information Technology
Monash University, Australia.
Email: {kaiming.ting,jonathan.wells}@monash.edu

***Abstract*—Mass estimation, an alternative to density estimation, has been shown recently to be an effective base modelling mechanism for three data mining tasks of regression, information retrieval and anomaly detection. This paper advances this work in two directions. First, we generalise the previously proposed one-dimensional mass estimation to multi-dimensional mass estimation, and significantly reduce the time complexity to $O(\psi h)$ from $O(\psi^h)$—making it feasible for a full range of generic problems. Second, we introduce the first clustering method based on mass—it is unique because it does not employ any distance or density measure. The structure of the new mass model enables different parts of a cluster to be identified and merged without expensive evaluations. The characteristics of the new clustering method are: (i) it can identify arbitrary-shape clusters; (ii) it is significantly faster than existing density-based or distance-based methods; and (iii) it is noise-tolerant.**

***Keywords*-Mass estimation; mass-based clustering.**

I. Introduction

Mass estimation [12] was introduced recently as a base modelling mechanism in data mining. It is as fundamental as density estimation which has been the bedrock for most data modelling methods for a wide range of tasks such as classification, clustering, and anomaly detection. Mass estimation possesses the following properties [12]:

> "(i) a mass distribution stipulates an ordering from core points to fringe points in a data cloud; (ii) this ordering accentuates the fringe points with a concave function—the essential property that is easily exploited by existing algorithms to improve their task-specific performance; and (iii) it is a constant-time-and-space-complexities estimation method."

Ting et al [12] show that models using mass estimation perform at least as good as and often better than a total of eight state-of-the-art methods in terms of task-specific performance measures in three tasks: information retrieval, regression and anomaly detection.

Notwithstanding its successful applications, the major limitation of existing mass estimation is that it is designed for one-dimensional problems only.

Our first contribution in this paper is to extend the above-mentioned work to a new version of mass estimation which allows it to

- remove the current limitation of one-dimensional mass estimation to enable multi-dimensional mass estimation;
- reduce the time complexity to $O(\psi h)$ from $O(\psi^h)$, making it feasible for very high level mass estimation, where ψ is the sampling size, and h is the level.

These enhancements boost its applicability to a full range of generic problems, unconstrained by one-dimensional applications and low level mass estimation.

While maintaining the three properties mentioned above, the new proposed mass estimation is capable of modelling arbitrary shapes in multi-dimensional data—the key weakness of one-dimensional mass estimation.

In our second contribution, we show that the mass model produced for mass estimation can be employed to perform clustering more efficiently and effectively than existing state-of-the-art methods. The first mass-based clustering method we introduce is distinguished from the existing density-based or distance-based clustering methods by

- Making no density or distance calculation which is the major expense in any density-based or distance-based methods;
- Exploiting the structure provided by the mass model to identify clusters without expensive evaluations. This significantly speeds up the clustering process, leading to a sublinear time complexity algorithm in average case.

In the next section, we reiterate the original one-dimensional mass estimation, as presented by [12]. Section III introduces our proposed multi-dimensional mass estimation. Section IV describes a special tree structure designed for the mass estimation, and Section V demonstrates the modelling capability of the proposed mass estimation method. We introduce the first mass-based clustering method in Section VI and present the evaluation result in the next section. The last two sections give further discussion, conclusions and possible future work. The key symbols and notations used are given in Table I.

II. One-dimensional mass estimation

Ting et al [12] introduce one-dimensional mass estimation $mass(x, h)$, where the level $h \geq 1$. $h = 1$ mass estimation guarantees concavity, independent of the underlying density distribution; and $h > 1$ is used to model multi-modal mass

Table I: Symbols and notations.

x	An one-dimensional instance in $\mathcal{R}$
$\mathbf{x}$	An instance in $\mathcal{R}^d$
D	A data set of $\mathbf{x}$, where $\|D\| = n$
$\mathcal{D}$	A subset of D, where $\|\mathcal{D}\| = \psi$
$\mathbf{z}$	An instance in $\mathcal{R}^t$
D'	A data set of $\mathbf{z}$
h	Level of mass estimation
$m_\imath(.)$	Mass base function defined using binary split $s_\imath$
$\mathbf{m}(.)$	Generalised mass base function
$T(.)$	A function which splits $\mathcal{R}^d$ into subspaces
$\overline{mass}(.)$	One-dimensional mass function which returns a real value
$\overline{gmass}(.)$	Multi-dimensional mass function which returns a real value
t	Number of models used in $\overline{mass}(.)$ or $\overline{gmass}(.)$

distribution. It is defined w.r.t. $D = \{x_1, x_2, \cdots, x_{n-1}, x_n\}$, where $x_1 < x_2 < \cdots < x_{n-1} < x_n$ on the real line.

$$mass(x,h) = \begin{cases} \sum_{\imath=1}^{n-1} mass_\imath(x, h\text{-}1)p(s_\imath), & h > 1 \\ \sum_{\imath=1}^{n-1} m_\imath(x)p(s_\imath), & h = 1 \end{cases} \quad (1)$$

The mass base function $m_\imath(x)$, as a result of a binary split at $s_\imath$ on the real line, is defined as

$$m_\imath(x) = \begin{cases} m_\imath^L = \imath & \text{if } x \le s_\imath \\ m_\imath^R = n - \imath & \text{if } x > s_\imath \end{cases}$$

L and R denote the left and right sides of the split, respectively; and $p(s_\imath) = (x_{\imath+1} - x_\imath)/(x_n - x_1) > 0$ is the probability of the binary split.

A high level mass estimation is computed recursively by using the mass estimations obtained at lower levels. A binary split at $s_\imath$ in a level-h (> 1) mass estimation produces two level-(h-1) mass estimations: (a) $mass_\imath^L(x, h\text{-}1)$—the mass estimation for $x \le s_\imath$ which is defined using $\{x_1, \ldots, x_\imath\}$; and (b) $mass_\imath^R(x, h\text{-}1)$—the mass estimation for $x > s_\imath$ which is defined using $\{x_{\imath+1}, \ldots, x_n\}$.

$$mass_\imath(x, h\text{-}1) = \begin{cases} mass_\imath^L(x, h\text{-}1) & \text{if } x \le s_\imath \\ mass_\imath^R(x, h\text{-}1) & \text{if } x > s_\imath \end{cases}$$

Further, $mass(x,h)$ can be approximated using sample subsets. The approximate mass estimation $\overline{mass}(x,h)$ for a point $x \in \mathcal{R}$, is defined w.r.t. $\mathcal{D} = \{x_1, \ldots, x_\psi\}$, where $\mathcal{D}$ is a random subset of D, and $\psi \ll |D|$, $h < \psi$.

$$\overline{mass}(x,h) = \frac{1}{t}\sum_{k=1}^{t} mass(x, h|\mathcal{D}_k) \quad (2)$$

Ting et al [12] show one way that this one-dimensional mass estimation can be applied to multi-dimensional problems by conducting an one-dimensional mapping of the original $\mathcal{R}^d$ space to new mass space $\mathcal{R}^t$, where $d \ll t$; $\mathbf{x} = [x^1, \ldots, x^d]$, $\mathbf{x} \in D$; $\mathbf{z} = [z^1, \ldots, z^t], \mathbf{z} \in D'$. This is done by randomly selecting a subset $\mathcal{D} \subset D$ and a dimension q from $[1, \ldots, d]$ and then it builds a mass model $mass(x^q, h|\mathcal{D})$. This is repeated t times; and the t mass models can then be used to map every $\mathbf{x} \in D$ to $\mathbf{z} \in D'$ in the new mass space.

Ting et al [12] show that four existing algorithms perform better in the mass space than in the original space in terms of task-specific measures in two tasks: regression and information retrieval.

Despite this successful application, it is recognised by [12] that a multi-dimensional mass mapping, rather than the one-dimensional mass mapping, can further widen its applications to a full range of generic problems. We introduce one way to achieve this aim in the next three sections.

III. Multi-dimensional mass estimation

Here we propose a way to generalise the one-dimensional mass estimation that eliminates the need to compute the probability of binary split, $p(s_\imath)$; and it gives rise to a randomised version of the above equations. It requires two functions. First, a function that generates different (random) subspaces covering each point in the feature space. This generalises the binary split into half-space splits or 2^h-space splits when h levels of half-space splits are used. Second, a generalised version of mass base function is used to define mass in a subspace. The formal definition follows.

Let $\mathbf{x}$ be an instance in $\mathcal{R}^d$. Let $T(\mathbf{x})$ be one of the two half-spaces in which $\mathbf{x}$ falls into; and m the number of training instances in the half-space.

Generalised mass base function: $\mathbf{m}(T(\mathbf{x}))$ is defined as

$$\mathbf{m}(T(\mathbf{x})) = \begin{cases} \mathsf{m} & \text{if } \mathbf{x} \text{ is in a half-space of } T, \\ 0 & \text{otherwise.} \end{cases}$$

In one-dimensional problems, let $T_i(x)$ be one of the two half-spaces in which x falls into; and $T_i^h(x)$ be one of the 2^h-spaces in which x falls into.

Equations (1) and (2) can now be approximated as follows:

$$\sum_{\imath=1}^{n-1} m_\imath(x)p(s_\imath) \approx \frac{1}{t}\sum_{i=1}^{t} \mathbf{m}(T_i(x)) \quad (3)$$

$$mass(x,h) \approx \frac{1}{t}\sum_{i=1}^{t} \mathbf{m}(T_i^h(x)) \quad (4)$$

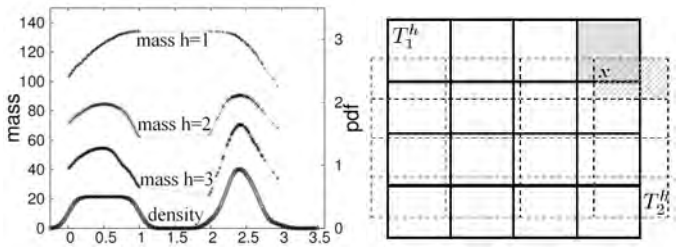
$$\overline{mass}(x,h) \approx \frac{1}{t}\sum_{i=1}^{t} \mathbf{m}(T_i^h(x|\mathcal{D}_i)) \quad (5)$$

Here every T_i (or T_i^h) is generated randomly with equal probability. Note that $p(s_\imath)$ in Equation (1) has the same assumption.

The new mass estimation for multi-dimensional problems is the same as Equation (5) by simply replacing x with $\mathbf{x}$:

$$\overline{gmass}(\mathbf{x}) \approx \frac{1}{t}\sum_{i=1}^{t} \mathbf{m}(T_i^h(\mathbf{x}|\mathcal{D}_i)) \quad (6)$$

Like its one-dimensional counterpart, the multi-dimensional mass estimation (i) stipulates an ordering from core points



(a) (b)

Figure 1: (a) An example of the new mass estimation for $h = 1, 2, 3$ in comparison with Gaussian kernel density estimation. (b) The result of two equal-size splits T_1^h and T_2^h implemented as two $h:d$-Trees using $h:d$ = 2:2. The two splits start with different working spaces—the two outer rectangles of solid and dotted boxes. The shaded cells are the subspaces in which x falls into in T_1^h and T_2^h, respectively.

(having high mass) to fringe points (having low mass) in a data cloud, regardless of its density distribution, including uniform density distribution; (ii) this ordering accentuates the fringe points with a concave function. An example of the new mass estimation is shown in Figure 1(a) for a uniform density distribution and a Gaussian density distribution.

IV. Mass estimation using $h:d$-Trees

$T^h(\mathbf{x}|\mathcal{D})$ can be implemented using a tree structure, where a working space is partitioned into 2^ℓ equal-size subspaces at the leaves of the tree with height $\ell = h \times d$, where d is the number of dimensions. Let m_j be the mass of subspace j; and there is a total of 2^ℓ subspaces which have a total mass: $|\mathcal{D}| = \sum_{j=1}^{2^\ell} \mathsf{m}_j$, where $\mathsf{m}_j = \mathbf{m}(T^h(\mathbf{x}|\mathcal{D}))$; and $\mathbf{x}$ is in subspace j of T^h.

We implement $T^h(\mathbf{x}|\mathcal{D})$ as an $h:d$-Tree, where each path from the root to a leaf has $h \times d$ nodes within which each of the d attributes appears exactly h times. For examples:

(a) $h:d$ = 1:2 generates a tree of height = 2 in a two-dimensional domain where each attribute is used exactly once on every path from the root to a leaf.

(b) $h:d$ = 3:4 generates a tree of height = 12 in a four-dimensional domain where each attribute is used three times on every path from the root to a leaf.

The procedure to generate an ensemble of $h:d$-Trees is given in Algorithm 1. Each $h:d$-Tree, an implementation of $T^h(\mathbf{x}|\mathcal{D})$, is generated after the following two randomisation processes:

(i) Each $\mathcal{D}$ of size ψ is a random sample of D. The sampling is conducted 'strictly' without replacement, i.e., $\mathbf{x} \neq \mathbf{y}$, where $\mathbf{x} \in \mathcal{D}_i, \mathbf{y} \in \mathcal{D}_j, \forall i, j$. This is done in line# 4 of Algorithm 1. The sampling process is restarted with D when the data run out.

(ii) Each **working space**, which covers $\mathcal{D}$, is initialised from a random perturbation as follows. For each attribute q, a split value (v_q) is chosen randomly within the range

Algorithm 1 : BuildTrees(D, t, ψ, h)

Inputs: D - input data, t - number of trees, ψ - sub-sampling size, h - number of times an attribute is employed in a path.
Output: F - a set of t $h:d$-Trees

1: $MaxHeightLimit \leftarrow h \times d$
2: **Initialize** F
3: **for** $i = 1$ to t **do**
4: $\quad \mathcal{D} \leftarrow sample(D, \psi)$ {strictly without replacement}
5: $\quad (min,max) \leftarrow$ InitialiseWorkingSpace($\mathcal{D}$)
6: $\quad F \leftarrow F \cup$ SingleTree$(\mathcal{D}, min, max, 0)$
7: **end for**

$[min_q(\mathcal{D}), max_q(\mathcal{D})]$, i.e., the minimum and maximum values of q in $\mathcal{D}$. Then, attribute q of the working space is defined having the range $[min_q, max_q] = [v_q - r, v_q + r]$, where $r = \max(v_q - min_q(\mathcal{D}), max_q(\mathcal{D}) - v_q)$. The ranges of all d dimensions define the working space used to generate an $h:d$-Tree. This is done in line# 5 of Algorithm 1.

Examples of the splits $T_1^h(\mathbf{x}|\mathcal{D}_1)$ and $T_2^h(\mathbf{x}|\mathcal{D}_2)$ using $h:d$ = 2:2, created from two different working spaces, are shown in Figure 1(b). The mass for each $\mathbf{x}$, estimated using Equation (6), is derived from t subspaces in which $\mathbf{x}$ falls into (see shaded cells in Figure 1(b).) Note that $\mathbf{m}(T^h(\mathbf{x}|\mathcal{D})) = 0$ if $\mathbf{x}$ falls outside the working space.

The tree building process is deterministic, shown in Algorithm 2. The $nextAttribute(A, \ell)$ routine (in line# 4) selects an attribute in a round-robin manner from the attribute set A as the height level ℓ increases. After an attribute is selected, the node is constructed by splitting the working space into two equal-volume half-spaces (line# 5-7). If the two half-spaces are non-empty, the process is then repeated recursively on each half-space (line# 17-20). The tree growing process stops when the height level limit is met (line# 2). If one of the two half-spaces is empty (therefore resulting a single-branch node), the ranges in the node defined in min and max is constrained accordingly without actually creating the single-branch node (line# 8-15.) This is to avoid creating unnecessary single-branch nodes—to reduce memory requirement.

Mass estimation using $h:d$-Trees. Once an ensemble of $h:d$-Trees is built, it is ready for mass estimation. For each instance $\mathbf{x}$, the number of instances of subspace j (of an $h:d$-Tree) in which $\mathbf{x}$ falls into is returned as mass:

$$h:d\text{-Tree}(\mathbf{x}) = \mathsf{m}_j = \mathbf{m}(T^h(\mathbf{x}|\mathcal{D}))$$

The multi-dimensional mass estimation can now be expressed as

$$\overline{gmass}(\mathbf{x}) \approx \frac{1}{t} \sum_{i=1}^{t} h:d\text{-Tree}_i(\mathbf{x}) \qquad (7)$$

Parameter discussion. We set $t = 1000$ to get a large ensemble. Then, only two other parameters need to be set: h and ψ. A guideline is given below.

Algorithm 2 : SingleTree($\mathcal{D}, min, max, \ell$)

Inputs: $\mathcal{D}$ - input data, min & max - arrays of minimum and maximum values for each attribute in A that defines a working space, ℓ - current height level, A - set of d attributes.
Output: an h:d-Tree

while (true) do
if ($\ell < MaxHeightLimit$) **then**
  {Retrieve an attribute from A based on height level.}
  $q \leftarrow nextAttribute(A, \ell)$
  $p \leftarrow (max_q + min_q)/2$
  $\mathcal{D}_l \leftarrow filter(\mathcal{D}, q < p)$
  $\mathcal{D}_r \leftarrow filter(\mathcal{D}, q \geq p)$
  if ($|\mathcal{D}_l| = 0$) or ($|\mathcal{D}_r| = 0$) **then**
    {Constrain range for single-branch node.}
    if ($|\mathcal{D}_l| > 0$) **then** $max_q \leftarrow p$
    else $min_q \leftarrow p$
    end if
    $\ell \leftarrow \ell + 1$
    continue at the start of while loop
  end if
  {Build two nodes: $Left$ and $Right$ as a result of a split into two equal-volume half-spaces.}
  $temp \leftarrow max_q$; $max_q \leftarrow p$
  $Left \leftarrow$ SingleTree($\mathcal{D}_l, min, max, \ell + 1$)
  $max_q \leftarrow temp$; $min_q \leftarrow p$
  $Right \leftarrow$ SingleTree($\mathcal{D}_r, min, max, \ell + 1$)
end if
terminate while loop
endwhile
return *Node*($Left, Right, SplitAtt \leftarrow q$, $SplitValue \leftarrow p, Size \leftarrow |\mathcal{D}|$)

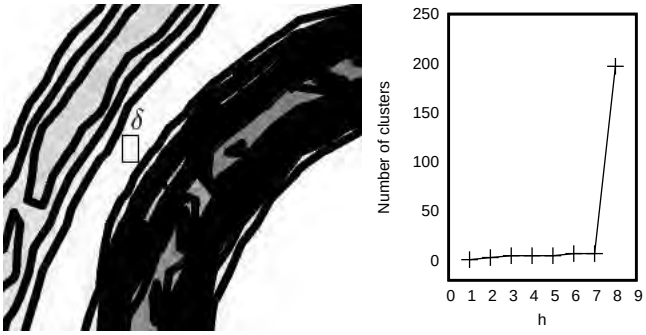
Setting h: The level setting influences the size of the subspaces in which mass is calculated. To differentiate one cluster from another, the zero-mass subspaces must be small enough to fit into the region separating different clusters—if the subspaces are too large, different clusters are joined.

Let δ be the minimum separation between any two clusters (in all dimensions); and r be the range of the working space in the same dimension δ is measured. In order to have zero-mass subspaces separating the two clusters, $\delta > \frac{r}{2^h}$. Thus,

$$h > log_2(\frac{r}{\delta})$$

Note that this setting is with reference to the ratio $\frac{r}{\delta}$ rather than the absolute separation δ. In other words, δ may be small, but if all the data is concentrated in a small area, then $\frac{r}{\delta}$ is small too; thus a low h can be used to separate these clusters with small δ. An example of separation required between two clusters is shown in Figure 2(a), where $\delta = 0.0077$ and $r = 0.97$ which yields $h > 6.97$.

Figure 2(b) shows how h can be set in practice (when used for clustering, to be described in Section VI): h is increased until a large number of clusters is identified. In this case, the setting before this large number shall be used, i.e., $h = 7$.



(a) (b)

Figure 2: (a) An example of δ in the horizontal dimension in the Ring-Curve data set (the full view of the data set is shown in the last diagram of Figure 4(b)). (b) As h increases from 1 to 8, the number of clusters identified by $\overline{gmass}(.)$ also increases as follows: 1,3,5,5,5,7,7,196 (in a 48-dimensional data set.)

Setting ψ: The complexity of the data influences the setting of ψ. While a large ψ is preferred, a small ψ can often be used. Simple clusters require small ψ; whereas complex clusters requires large ψ. Note that h also affects the choice of ψ. A high h should accompany a high ψ to avoid producing many isolated small islands in a cluster (when a low ψ is used.)

A. Time and Space Complexities

Using h:d-Tree reduces the time complexity to $O(min(hd, log\psi)\psi t)$ from $O(\psi^h\, t)$ (using Equation (2)), making it feasible for very high level-h mass estimation. The space complexity of h:d-Tree is $O(min(\psi, hd)t)$. It has almost the same order to $O(\psi t)$ when one-dimensional mass estimation using Equation (2) is stored using a lookup table [12].

B. Relation to k-d Tree

At the algorithmic level, a k-d Tree [6], based on median split, may appear on the surface to be similar to the h:d-Tree. For example, constructing a node of a k-d Tree starts on one dimension, and cycles through the dimensions to build subsequent nodes in the tree. h:d-Tree does a similar dimension cycling. However, there are important differences. First, the purposes differ: k-d Tree is designed to speed up search, e.g., in a near neighbour search; whereas h:d-Tree is specifically designed for mass estimation. Second, k-d Tree employs the median as the split point; in contrast, the split point for a node of h:d-Tree is simply the mid-point of a dimension in the working space, independent of the distribution of the data—no search is required to find the split point. Third, a k-d Tree cannot be used to estimate mass because the median-split produces a balanced tree with all external nodes having the same mass—useless

for our purpose! Fourth, a k-d Tree is constructed using all available data; whereas each h:d-Tree requires a small training sample only. As a result, although both are linear time-complexity algorithms, k-d Tree is linear with respect to the total training set size n; and h:d-Tree is linear with respect to $\psi \ll n$.

C. Mass's relation to density

Density is defined as mass per volume; whereas mass is defined independent of volume. Density is equivalent to mass only if the volume is the same for every subspace in a single tree. Because multiple trees are employed, the mass estimated by $\overline{gmass}(.)$ is not equivalent to density since the volume of each subspace varies from one tree to the next (see subspaces created by the two example trees in Figure 1(b).)

The only reason for using equal-size-subspace in each h:d-Tree is for fast tree construction—each division can be done without looking at the data, once the working space is defined. In fact, there is already a tree implementation that produces varying-size grid [9] which is designed specifically for anomaly detection only.

Further distinctions between mass estimation and density estimation have been provided by [12]. Their effects on clustering algorithms are given in Section VIII.

V. Modelling data distribution

Here we compare the one-dimensional mass estimation with the multi-dimensional mass estimation using h:d-Trees, in terms of their runtime and ability to model data distribution.

Runtime. Figure 3 shows the factor of increase in runtime when h increases from 1 to 5, when both mass estimations use $t = 1000$ and $\psi = 8$ in the Ring-Curve data set. The one-dimensional mass estimation using Equation (2) increases the runtime by a factor of 264 when increasing h from 1 to 5; whereas h:d-Trees's runtime increases by a factor of 1.07 only. Setting $\psi = 256$ and $h = 5$, the one-dimensional mass estimation fails to complete the run in less than one day, whereas h:d-Trees completes it in 130 seconds.

Because the one-dimensional mass estimation using Equation (2) is so slow and h:d-Trees can be easily modified to produce one-dimensional mass estimation (by randomly selecting an attribute to form A in Algorithm 2), we show the result of one-dimensional h:d-Trees instead in the following experiments. The one-dimensional mapping applied to multi-dimensional problems is the same as that described in Section II or [12].

Modelling the underlying data distribution for multi-dimensional problems. After h:d-Trees are trained using the given data, the modelling results are presented as contour maps, produced from the mass values estimated from h:d-Trees for a lattice of equal-spaced points in the entire feature space.

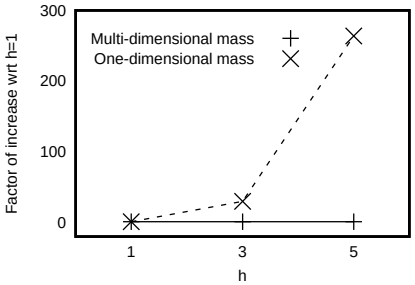


Figure 3: Runtime comparison: one-dimensional mass estimation versus multi-dimensional mass estimation.

Figure 4 shows the contour maps of the two mass estimations using a two-dimensional data set: Ring-Curve. They show the modelling progression as h increases from 2 to 6. The one-dimensional mass estimation, in Figure 4(a), fails to model the two clusters. In contrast, the multi-dimensional mass estimation successfully models the two clusters when $h = 6$ (see the last diagram in Figure 4(b).)

Figure 5 shows the key weaknesses of one-dimensional mass estimation using three additional data sets: one-Gaussian, three-Gaussian, and Ring. Although it models the uni-modal data reasonably well, it does poorly in the multi-modal data—it creates phantom modes (e.g., in the three-Gaussian and Ring data sets) because it assumes data symmetry. Multi-dimensional mass estimation does not have this kind of weakness and models the underlying data distribution well, when an appropriate h is used.

Because h:d-Trees has done the hard work to model the underlying data distribution, to identify clusters within the data is simply to extract the connecting structures from h:d-Trees. We will describe how this can be done in the next section.

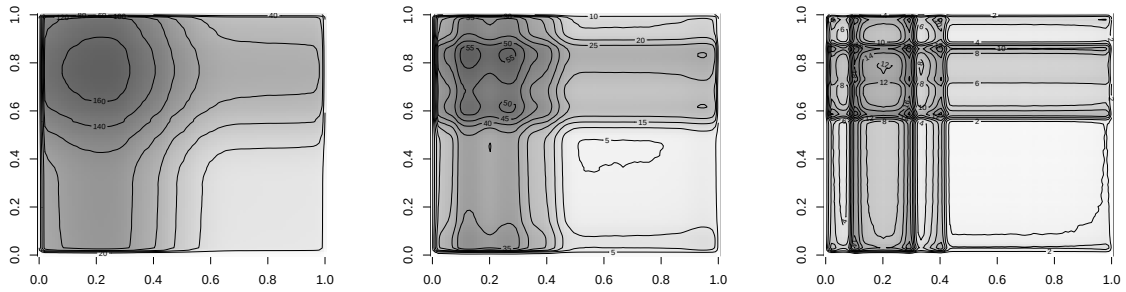
VI. Mass-Based Clustering

In this section, we show how to employ mass to perform clustering. It is unique among existing clustering methods because it does not employ any distance or density measure. Mass-based clustering has the following characteristics:

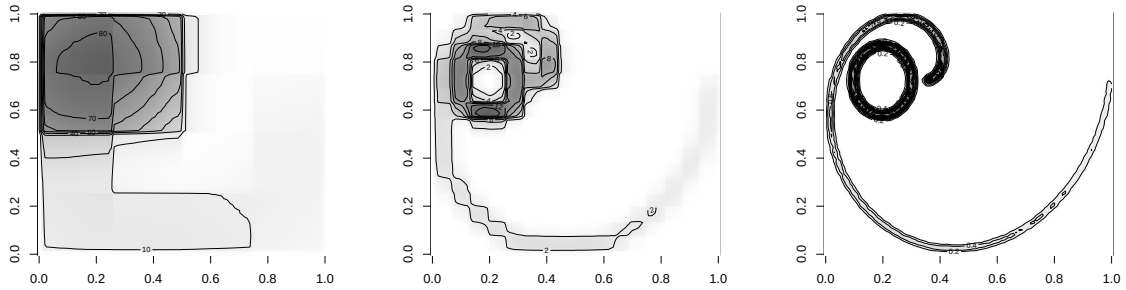
- It employs the same mass model to identify arbitrary-shape clusters and to filter noise;
- It does not assume any data distribution;
- It scales up well to huge data size;
- It performs clustering without the need to invoke distance or density calculations.

The local neighbourhood of any instance is readily available in h:d-Trees after they have been built. Thus, clustering based on mass gets the local neighbourhood information without additional cost.

We provide the definitions and the algorithm for mass-based clustering in the next two sections, and the time and space complexities in the third section.

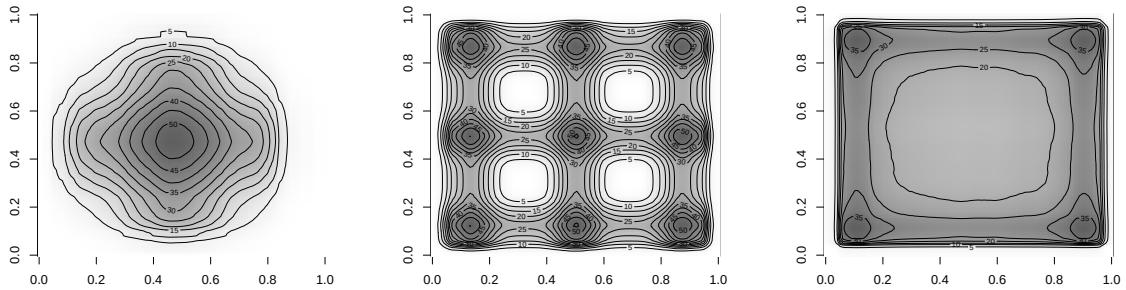


(a) One-dimensional mass estimation, with $h = 2, 4, 6$

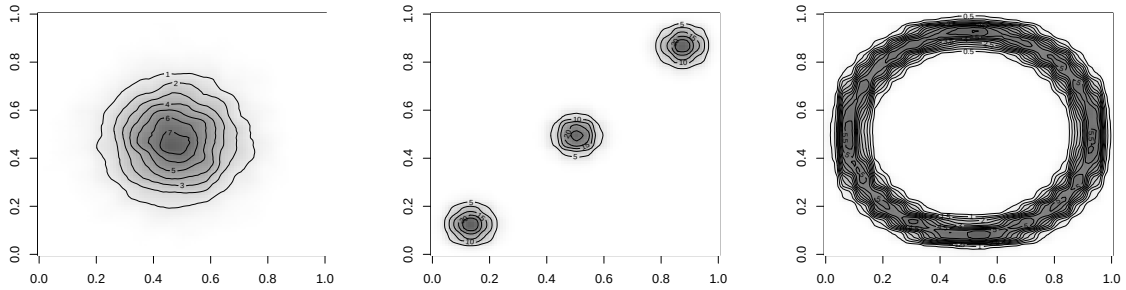


(b) Multi-dimensional mass estimation, with $h = 2, 4, 6$

Figure 4: Contour maps from mass estimation using h:d-Trees ($\psi = 256$, $t = 1000$) for the Ring-Curve data set.



(a) One-dimensional mass estimation



(b) Multi-dimensional mass estimation

Figure 5: Contour maps from mass estimation using h:d-Trees ($\psi = 256$, $t = 1000$, $h = 4$) for the one-Gaussian, three-Gaussian, and Ring data sets, shown in the first, second and third column, respectively.

A. Definitions

To simplify notation, we use T to denote a subspace defined by $T(\mathbf{x})$ in which an instance $\mathbf{x}$ falls into.

Definition 1: An instance $\hat{\mathbf{x}} \in T_i \subset \mathcal{R}^d$ is a **seed** to form a cluster, iff $\mathbf{m}(T_i(\hat{\mathbf{x}})) > 0$.

Recall that $T_i(.)$ is built and $\mathbf{m}(T_i(.))$ is estimated using $\mathcal{D}_i \subset D$ (see Equation (6)), where $|\mathcal{D}_i| \ll |D|$.

Definition 2: An instance $\mathbf{x} \in D$ is T-**connected** by a seed $\hat{\mathbf{x}} \in T_i$, iff
(a) $\mathbf{x} \in T_i$, or
(b) $\mathbf{x} \in T_{I_{j+1}}$, $\mathbf{x} \notin (T_i \cup (\cup_j T_{I_j}))$: $\mathbf{m}(T_i \cap T_{I_1}) > 0$ and $\mathbf{m}(T_{I_1} \cap T_{I_2}) > 0$ and ... and $\mathbf{m}(T_{I_j} \cap T_{I_{j+1}}) > 0$, where I_j is an index and $j \geq 1$.

Definition 3: A **seed-based cluster** is a subset $C \subseteq D$, where $\forall \mathbf{x} \in C : \mathbf{x}$ is T-connected by $\hat{\mathbf{x}}$.

Definition 4: A **cluster-pair** $[g, h]$ consists of two seed-based clusters with $\hat{\mathbf{x}}_g$ and $\hat{\mathbf{x}}_h$, where $\exists \mathbf{x} \in D : \mathbf{x}$ is T-connected jointly by $\hat{\mathbf{x}}_g$ and $\hat{\mathbf{x}}_h$.

Definition 5: An **arbitrary-shape cluster**, for the set of seeds S and the set of cluster-pairs P, is a subset $C \subseteq D$, iff
(a) $\forall \mathbf{x} \in C,\ \exists \hat{\mathbf{x}} \in S : \mathbf{x}$ is T-connected by $\hat{\mathbf{x}}$, and
(b) $\forall \hat{\mathbf{x}}_g, \hat{\mathbf{x}}_h \in S$, either $[g, h] \in P$ or $\exists \hat{\mathbf{x}}_{F_1}, \ldots, \hat{\mathbf{x}}_{F_\imath} \in S$: $\{[g, F_1], [F_1, F_2], \ldots, [F_\imath, h]\} \subseteq P$, where $F_\imath$ is an index and $\imath \geq 1$.

Definition 6: **Noise instances** are instances in C where $|C| < \eta$, where η is a constant defined by users.

We call the process to find arbitrary-shape clusters from a given data set and h:d-Trees, a T-**connection process**.

It is interesting to note that one may invoke a search to find a seed which is either the centre of the cluster (i.e., the instance having the highest mass, $\max_g \overline{gmass}(\hat{\mathbf{x}}_g)$) or $\hat{\mathbf{x}} \in T_i$ having the highest mass, $\max_i \mathbf{m}(T_i(\hat{\mathbf{x}}))$. But, the clustering result obtained is exactly the same as that obtained by any seed satisfying Definition 1. This is because the T-connection process will eventually encompass the entire cluster, no matter which instance in the cluster is chosen as the seed. We employ the cheapest, no-search option in formulating the mass-based clustering algorithm called MassTER in the next section.

B. The MassTER Algorithm

The idea is to extract the connecting structures from different h:d-Trees—which have already modelled the underlying data distribution—in order to identify arbitrary-shape clusters that are free of noise instances, using the definitions provided in the last section. Recall that each h:d-Tree is a realisation of T_i. The algorithm aims at uniquely assigning every instance to one cluster.

The mass-based clustering procedure, MassTER, is given in Algorithm 3. The first step is to build an ensemble of h:d-Trees in order to obtain the connecting subspaces T_i. The second step is to assign a cluster to each instance if T_i has previously been labelled with a cluster ID (Definition 2.) If T_i is labelled, this step also checks whether the instance is T-connected to another seed-based cluster (Definition 4). If it is, a cluster-pair is formed. If T_i is unlabelled, the instance is designated as the seed of a new cluster (if it satisfies Definition 1) and T_i is labelled with the same new cluster ID. The third step is to merge all cluster-pairs, that satisfy Definition 5(b), into a single arbitrary-shape cluster.

Algorithm 3 : MassTER(D, t, ψ, h, η)

Inputs: D - input data, t - number of trees, ψ - sub-sampling size, h - number of times an attribute is employed in a path, η - minimum number of instances in a cluster.

1: $\{T_i : i = 1, \ldots, t\} \leftarrow$ BuildTrees(D, t, ψ, h)
2: Assign a seed-based cluster to each instance $\mathbf{x} \in D$ (satisfying Definitions 1 or 2) and identify all cluster-pairs satisfying Definition 4.
3: Merge cluster-pairs which satisfy Definition 5(b).
4: $E \leftarrow$ clusters having number of instances less than η.
5: **return** K arbitrary-shape clusters, $C_\jmath, \jmath = 1, \ldots, K$; and noise instances in E.

An example of the second step is shown in Figure 6(a). Assume $\hat{\mathbf{x}}_g, \hat{\mathbf{x}}_h, \mathbf{x}_1, \mathbf{x}_2$ are assigned a cluster ID in the following sequence, given $T_1, \ldots, T_5$ and $\forall i, \mathbf{m}(T_i(.)) > 0$:

$\hat{\mathbf{x}}_g$: Since this is the first instance to be assigned a cluster, no T_i has label. Thus, $\hat{\mathbf{x}}_g$ is designated as the seed, and T_1, T_2, T_3 and $\hat{\mathbf{x}}_g$ are labelled with a new cluster ID: g (Definition 1.)

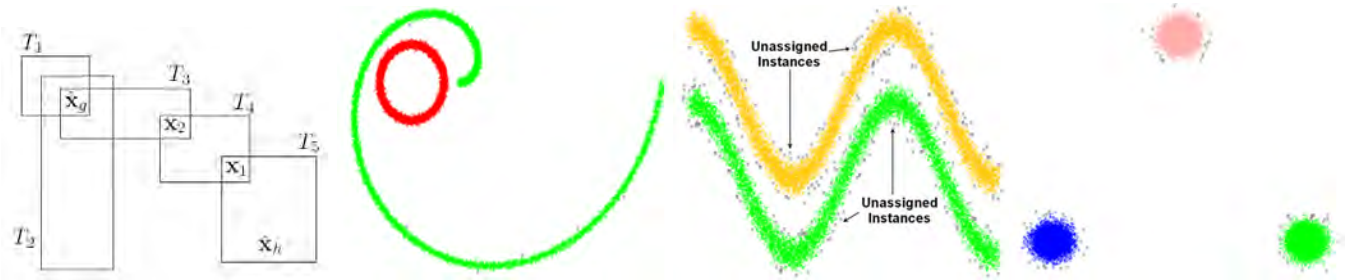
$\hat{\mathbf{x}}_h$: Since T_5 is unlabelled at this point in time, $\hat{\mathbf{x}}_h$ is designated as the seed, and T_5 and $\hat{\mathbf{x}}_h$ are labelled with a new cluster ID: h (Definition 1.)

$\mathbf{x}_1$: T_5 has a label now, $\mathbf{x}_1$ is assigned with the cluster ID: h. T_4 is also labelled with the same ID (Definition 2.)

$\mathbf{x}_2$: Since both T_3 and T_4 have been labelled with different cluster IDs, a cluster-pair is formed: $[g, h]$ (Definition 4.)

Note that the second step produces different results for different orderings, e.g., the reverse ordering of the above example will assign all four instances to a single cluster ID. However, the merging process will yield the same clustering result, independent of the ordering of the instances.

At the end of the merging process, all clusters having the number of instances less than η are considered as noise instances; they are filtered out in the fourth step of the algorithm. η is the only additional parameter in MassTER, and it is set to 10 in our experiments—a cluster of less than 10 instances is too small to be considered as a proper cluster.



(a) Example assignment of cluster ID (b) Ring-Curve (c) Wave (d) Triangular-Gaussian

Figure 6: (a) An example of assigning a cluster ID to each of the four instances in step 2 of Algorithm 3. (b)-(d) Scatter plots of the clustering result by MassTER in the 3-dimensional Ring-Curve-Wave-TriGaussian data set. Note that all unassigned instances are at the periphery of each cluster.

C. Time and space complexities

The time complexity analysis is as follows. The construction of h:d-Trees costs $O(t\psi hd)$ (assume $hd > log(\psi)$). The most expensive part of the clustering procedure is to assign a seed cluster to each instance which costs $O(tnhd)$. The last two steps cost $O(n)$ in the worse case. Thus, the overall time cost for the MassTER algorithm is $O(tnhd)$, as $\psi \ll n$. The main space requirement is to store h:d-Trees and input D; the space requirements in other steps are substantially lower. Thus, MassTER has space complexity $O(thd + n)$ during training. The training set is discarded after training, yielding $O(thd)$.

VII. Empirical Evaluation

We use DBSCAN [5] and DENCLUE [3] as the benchmarks because they both claim to be fast running density-based clustering algorithms. MassTER is implemented in JAVA, and we use DBSCAN in WEKA [13] and a version of DENCLUE implemented in R (www.r-project.org) in our empirical evaluation. The experiments are run on a machine having 2 $\times$ Xeon X5550 Quad-core 2.66 GHz processors and 48GB memory (www.vpac.org).

The clustering result is reported in terms of CPU runtime (in seconds), number of clusters identified, number of unassigned instances, and F-measure which is calculated based on assigned instances only. F-measure = 1 when all assigned instances are in the correct clusters, i.e., perfect clustering; and F-measure = 0 if all instances are assigned to wrong clusters. We tune the parameters of each algorithm and report the best result.

We use five data sets that have the characteristics in which clustering methods can be evaluated: clusters of different shapes, sizes and densities; for example, one data set has clusters embedded in high dimensional space; one has significant amount of noise; and one has overlapping clusters. We describe the experimental result with each data set in the following subsections.

Table II: Clustering results in the Ring-Curve-Wave-Tri-Gaussian data sets for MassTER ($h = 7$ and $\psi = 256$) and DBSCAN ($\epsilon = 0.01$ and $minPts = 6$).

	3-dimensional data		48-dimensional data	
	MassTER	DBSCAN	MassTER	DBSCAN
Runtime	59	1357	256	12021
#cluster [7]	7	8	7	8
#unassigned	321	332	692	332
F-measure	1.0000	0.9999	1.0000	0.9999

Ring-Curve-Wave-Tri-Gaussian. It has three two-dimensional synthetic data embedded in either a 3-dimensional data set or a 48-dimensional data set (where 42 dimensions are irrelevant with a constant value.) The three two-dimensional data are Ring-Curve, Wave and Triangular-Gaussian shown in Figure 6(b),(c),(d), which have a total of seven clusters. Each cluster has 10000 instances with a total of 70000 instances.

The clustering results from MassTER and DBSCAN are shown in Table II. In both the 3-dimensional and 48-dimensional data sets, MassTER performs better than DBSCAN in three out of the four performance measures: MassTER run faster than DBSCAN by a factor of 23 and 47, respectively in the two data sets; MassTER identifies the correct 7 (versus 8) clusters and the perfect F-measure of 1 (versus 0.9999). MassTER has increased its number of unassigned instances from 321 to 692 when the number of dimensions increases from 3 to 48; whereas DBSCAN has the same 332 unassigned instances in both cases. Note that MassTER can reduce the number of unassigned instances by increasing ψ, e.g., setting ψ from 256 to 2560 reduces the number of unassigned instances from 692 to 317; and this only increases the runtime from 256 to 646 seconds.

The unassigned instances by MassTER are all at the periphery of each cluster shown in Figure 6(b),(c),(d).

In order to examine how well the algorithms scale up, we use the 48-dimensional data set and increase the data size from 7000 to 70000, 525000, and 1050000. Figure 7 plots runtime ratio versus data size ratio (1, 10, 75 and 150) by using 7000 as the base. The result shows that MassTER has

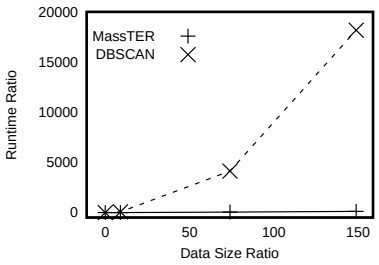


Figure 7: Runtime scale up comparison: MassTER vs DBSCAN in the 48-dimensional Ring-Curve-Wave-TriGaussian data set. Note that DBSCAN completes the task of the one-million data set (at data size ratio=150) in 36 days versus MassTER's 1.3 hours.

a sublinear increase in runtime: The runtime ratio increases from 1 to 112 when the data size ratio increases from 1 to 150. In contrast, DBSCAN's runtime ratio increases from 1 to more than 18000 with the same increase in data size ratio. MassTER is faster than DBSCAN by a factor of 655 when the one-million data set is used.

VaryingDensity. This data set has 2 dimensions and 2 clusters of different densities, and each cluster has 250 instances. The two clusters are two Gaussian distributions with density ratio $dr = \frac{\sigma_2}{\sigma_1}$ ranging from 30 to 50 where cluster 1 has fixed σ_1=1. At dr=50, dense cluster 1 is within the boundary of sparse cluster 2. The result shown in Table III reveals that MassTER produces better clustering than DBSCAN in terms of F-measure and number of clusters. This result shows that MassTER is more tolerant to varying densities than DBSCAN. In this small data set of 500, DBSCAN runs faster than MassTER because MassTER has some fixed overhead, independent of the data size.

Table III: Clustering results in VaryingDensity data sets for MassTER ($h = 5$) and DBSCAN ($minPts = 6$).

	$dr = 30$		$dr = 50$	
	MassTER $\psi = 256$	DBSCAN $\epsilon = 0.069$	MassTER $\psi = 500$	DBSCAN $\epsilon = 0.053$
Runtime	1.1	0.1	1.8	0.2
#cluster [2]	2	2	2	3
#unassigned	47	34	59	55
F-measure	1.000	0.983	0.995	0.962

OneBig. This data set was previously employed by [11]. It has 20 attributes and 9 clusters. The biggest cluster has 50,011 instances, and each of the other eight clusters has approximately 1000 instances. In addition, there are 10,000 noise instances randomly distributed in the feature space. This data set has a total of 68,000 instances.

The result in Table IV shows that MassTER and DBSCAN have same clustering result in terms of F-measure and number of clusters; but MassTER runs significantly faster than DBSCAN is this large data set. Note that both MassTER and DBSCAN have correctly identified the 10000 noise instances in this data set.

Table IV: Clustering results in the OneBig data set for MassTER ($h = 3$ and $\psi = 256$) and DBSCAN ($\epsilon = 0.1$ and $minPts = 6$).

	MassTER	DBSCAN
Runtime	146	6738
#cluster [9]	9	9
#unassigned	10022	10005
F-measure	1.00	1.00

Iris and Yeast. Iris has 150 instances, 4 dimensions and 3 clusters; Yeast has 1484 instances, 8 dimensions and 10 clusters. The clustering results are presented in Table V. In Iris, MassTER performs the best in terms of #cluster and F-measure though slower than DBSCAN (for this small data set). It is suspected that Yeast has significant overlapping clusters; this is reflected in low F-measure in all methods. It appears that all three methods have problems with overlapping clusters.

Table V: Clustering results in Iris and Yeast. The settings used for Iris are: MassTER ($h = 4, \psi = 64$), DBSCAN ($\epsilon = 0.1, minPts = 5$), DENCLUE ($tol = 0.0001, ctol = 1, b = 0.3$). Yeast: MassTER ($h = 3, \psi = 16$), DBSCAN ($\epsilon = 0.07, minPts = 5$), DENCLUE ($tol = 0.1, ctol = 70000, b = 0.06$)

	Iris (3 clusters)			Yeast (10 clusters)		
	Mass	DBSC	DENC	Mass	DBSC	DENC
Runtime	1.1	0.1	58.8	1.4	1.8	40.9
#cluster	3	5	4	11	12	10
#unassigned	58	70	0	966	1197	0
F-measure	1.00	0.89	0.88	0.34	0.2	0.08

Note that DENCLUE is very sensitive to parameter settings; Han and Kamber [7] have also reported the same observation. Because DENCLUE is significantly slower than either MassTER or DBSCAN, it cannot complete the execution in reasonable time for the large data sets we have presented earlier.

VIII. Discussion

For the purpose of clustering, the use of mass estimation avoids the key weakness of using density estimation: It is computationally expensive to get accurate density estimation. This is why DENCLUE [3] has to use grid instead for practical applications.

Mass has the following advantages over density. First, MassTER can use any instance of a cluster as the seed to form a cluster; DENCLUE relies on an density-attractor to form a cluster—this requires a search for a local maxima in the density estimation function. Though a hill-climbing search is employed (and a further improved search is described in [4]), the search is still a considerable computational expense that MassTER does not need. Second, constructing the grid takes $O(nlogn)$ for DENCLUE when storing the grid in a tree-structure. In contrast, MassTER

takes only $O(1)$ to construct the trees because all the parameters in $O(min(hd, log\psi)\psi t)$ are constant.

Each hyper-sphere centred at $\mathbf{x}$ in DBSCAN corresponds with a subspace T in MassTER—that is the key difference between the two algorithms. The subsequent steps to find clusters depend on this first step. Thus, the difference boils down to density versus mass—DBSCAN takes $O(n^2)$ to compute density; MassTER takes $O(n)$ to compute mass.

Most subspace clustering methods [8] are either bottom-up or top-down algorithms. Bottom-up algorithms (e.g., [10]) first examine one dimensional projections and then increasingly higher dimensions. Top-down algorithms (e.g., [2]) examine all dimensions and then assess the local neighbourhood to determine the best subspaces to identify clusters. MassTER is a top-down algorithm but it avoids the key pitfall of existing top-down algorithms: high computational cost with worse case time complexity $O(d^2)$ because of distance calculations. MassTER has used the T-connection process to assess local neighbourhood encapsulated in the mass model, without the need to compute distance.

It is interesting to note that MassTER completes a clustering task without computing mass using Equation (7), except checking $\mathbf{m}(T(.)) > 0$. When the structure of the clusters discovered is required, mass distribution can be computed with a minimum cost from h:d-Trees for each cluster—this provides useful information about the structure of the cluster, e.g., the peak(s) or centre(s) of the cluster, and the distribution of the data over the entire cluster, shown as contour maps in Figures 4 and 5.

IX. Conclusion and future work

This paper advances the first work on mass estimation in two significant ways. First, without the version we introduce here, existing mass estimation can only apply to one-dimensional problems and is limited to low level-h mass estimation because of its high time-complexity $O(\psi^h)$. We show that the new mass estimation can model arbitrary-shape distributions in multi-dimensional problems, and has time-complexity $O(\psi h)$ only.

Second, we realise one potential of mass, as a base modelling mechanism, to solve different kinds of data mining problems. The mass-based clustering method we introduced demonstrates that mass can be employed as an alternative to distance or density—the commonly used measures to design data mining methods. This is an example of applying mass directly to solve problems that yields a net gain in efficacy and efficiency. The proposed method identifies each cluster through a T-connection process by examining the structure of the mass model, without expensive evaluations. The result is an efficient noise-tolerant clustering method which can identify arbitrary-shape clusters. It has average case sublinear time complexity and linear space complexity w.r.t. input size; and it is shown to run significantly faster than existing density-based methods DBSCAN and DENCLUE.

In the near future, we will investigate how to apply the multi-dimensional mass estimation to a host of tasks such as classification and regression. In clustering, we will explore the ability of the mass-based clustering method in dealing with high dimensional problems and overlapping clusters.

Acknowledgment

This work is partially supported by the Air Force Research Laboratory, under agreement# FA2386-10-1-4052. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Hiroshi Motoda, Zhouyu Fu, Peter Tischer, Ray Smith and the anonymous reviewers have provided many helpful comments.

References

[1] R. Agrawal, J. Gehrke, D. Gunopulos and P. Raghavan, Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications, Proceedings of SIGMOD, 1998, 94–105.

[2] C. Böhm, K. Kailing, H.-P. Kriegel and P. Kröger, Density connected clustering with local subspace preferences, Proceedings of IEEE ICDM, 2004, 27-34.

[3] A. Hinneburg and D. A. Keim, An Efficient Approach to Clustering in Large Multimedia Databases with Noise, Proceedings of KDD, 1998, 58–65, AAAI Press.

[4] A. Hinneburg and H.-H. Gabriel, DENCLUE 2.0: Fast Clustering Based on Kernel Density Estimation, Proceedings of Intelligent Data Analysis, 2007, 70–80.

[5] M. Ester H.-P. Kriegel J. Sander and X. Xu, A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, Proceedings of KDD, 1996, 226–231.

[6] J.H. Friedman, J.L. Bentley and R.A. Finkel, An Algorithm for finding best matches in logarithmic expected time, ACM Transactions on Mathematical Software, 3, 3, 1997, 209-266.

[7] J. Han and M. Kamber, Data Mining: Concepts and Techniques, Second Edition, 2006, Morgan Kaufmann.

[8] H.-P. Kriegel P. Kröger and A. Zimek, Clustering High-Dimensional Data: A Survey on Subspace Clustering, Pattern-Based Clustering, and Correlation Clustering, ACM Transactions on Knowledge Discovery from Data, 3, 1, 2009, 1-58.

[9] F.T. Liu, K.M. Ting and Z.-H. Zhou, Isolation Forest, Proceedings of IEEE ICDM, 2008, 413–422.

[10] G. Moise, J. Sander and M. Ester, P3C: A Robust Projected Clustering Algorithm, Proceedings of ICDM, 2006, 414–425.

[11] A. Nanopoulos and Y. Theodoridis and Y. Manolopoulos, Indexed-based density biased sampling for clustering applications, IEEE Transaction on Data and Knowledge Engineering, 57, 1, 2006, 37-63.

[12] K.M. Ting, G.-T. Zhou, F.T. Liu and J.S.C. Tan, Mass Estimation and Its Applications, Proceedings of SIGKDD, 2010, 989-998.

[13] I.H. Witten and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, 2005, Morgan Kaufmann.

Density Estimation based on Mass

Kai Ming Ting*, Takashi Washio†, Jonathan R. Wells* and Fei Tony Liu*
*Gippsland School of Information Technology, Monash University,
Gippsland Campus, Churchill, Victoria 3842, Australia
Email: {kaiming.ting, jonathan.wells, tony.liu}@monash.edu
†The Institute of Scientific and Industrial Research, Osaka University,
8-1 Mihogaoka, Ibarakishi, Osaka, 5670047, Japan.
Email: washio@ar.sanken.osaka-u.ac.jp

***Abstract*—Density estimation is the ubiquitous base modelling mechanism employed for many tasks such as clustering, classification, anomaly detection and information retrieval. Commonly used density estimation methods such as kernel density estimator and k-nearest neighbour density estimator have high time and space complexities which render them inapplicable in problems with large data size and even a moderate number of dimensions. This weakness sets the fundamental limit in existing algorithms for all these tasks.**

We propose the first density estimation method which stretches this fundamental limit to an extent that dealing with millions of data can now be done easily and quickly. We analyze the error of the new estimation (from the true density) using a bias-variance analysis. We then perform an empirical evaluation of the proposed method by replacing existing density estimators with the new one in two current density-based algorithms, namely, DBSCAN and LOF. The results show that the new density estimation method significantly improves the runtime of DBSCAN and LOF, while maintaining or improving their task-specific performances in clustering and anomaly detection, respectively. The new method empowers these algorithms, currently limited to small data size only, to process very large databases — setting a new benchmark for what density-based algorithms can achieve.

***Keywords*-density estimation; density-based algorithms;**

I. Introduction

Density estimation is ubiquitously applied to various tasks such as clustering, classification, anomaly detection and information retrieval. Despite its pervasive use ('estimation of densities is a universal problem of statistics' [18]), there are no efficient density estimation methods thus far. Most existing methods such as kernel density estimator and k-nearest neighbour (k-NN) density estimator cannot be applied to problems with even a moderate number of dimensions and large data size. This paper is motivated to introduce the first efficient method for density estimation. We show that two existing density-based algorithms, which employ the new density estimator, set a new runtime benchmark that is orders of magnitude faster. For example, these two algorithms now take only days instead of months to complete tasks involving millions of instances, after the existing density estimators are replaced with the new one.

We make four contributions in this paper:

1) Propose a new density estimation method which has a significant advantage over existing methods in terms of time and space complexities.
2) Establish the characteristics of the method through a bias-variance analysis of the error of the new method.
3) Verify the generality of the method by replacing existing density estimators with the new one in two current density-based algorithms.
4) Significantly simplify and speed up the current algorithms using set-based definitions instead of the common point-based definitions (see Section V.)

The new density estimation method distinguishes itself from existing methods by:

- Employing no distance measures in the density estimation process.
- Having average case sublinear time complexity and constant space complexity. Thus, it can be applied to very large databases in which current methods such as kernel and k-NN density estimators are infeasible because they are prohibitively expensive to compute.

Two existing density estimators are presented in Section II, in order to contrast with the new density estimator we introduce in Section III. We analyze the error produced by the new estimator by a bias-variance analysis and provide a comparison of the estimation results between the new estimator and kernel density estimator in Section IV. Sections V and VI describe how the new estimator can replace existing density estimators in two current state-of-the-art density-based algorithms and their empirical evaluation results, respectively. A discussion of the related issues and the conclusions are provided in the last two sections.

II. Density Estimation

This section describes two, probably the most commonly used, density estimation methods, namely kernel density estimator and k-nearest neighbour density estimator.

A. Kernel Density Estimator

Let $\mathbf{x}$ be an instance in a d-dimensional space $\mathcal{R}^d$. The kernel density estimator (KDE) defined by a kernel function $K(\cdot)$ and bandwidth b is given as follows [13].

$$\bar{f}_{KDE}(\mathbf{x}) = \frac{1}{nb^d} \sum_{i=1}^{n} K(\frac{\mathbf{x} - \mathbf{x}_i}{b})$$

The difference $\mathbf{x} - \mathbf{x}_i$ requires some form of distance measure; and n is the number of instances in the given data set D. An example of $K(\cdot)$, as a rectangular function, is given as follows.

$$K(\mathbf{x}) = \begin{cases} \frac{1}{2} & \text{if } |\mathbf{x}| < 1 \\ 0 & \text{otherwise.} \end{cases}$$

B. k-NN Density Estimator

A k-nearest neighbour (k-NN) density estimator can be expressed as follows [14].

$$\bar{f}_{kNN}(\mathbf{x}) = \frac{|N(\mathbf{x}, k)|}{n \sum\limits_{\mathbf{x}' \in N(\mathbf{x},k)} distance(\mathbf{x}, \mathbf{x}')}$$

where $N(\mathbf{x}, k)$ is the set of k nearest neighbours to $\mathbf{x}$; and the search for nearest neighbours is conducted over D of size n.

Both KDE and k-NN density estimators have $O(n^2)$ time complexity and $O(n)$ space complexity in order to estimate the densities of n instances. Although there are various indexing schemes to speed up the search for nearest neighbour in order to aid the k-NN density estimator, they are not satisfactory in terms of dealing with high dimensional problems and large data sets. We will provide further discussion of this issue in Section VII.

III. Density estimator based on mass

A recently introduced base measure called mass [17] has demonstrated its wide application to solve various data mining tasks such as regression, information retrieval, clustering and anomaly detection, including one in data stream [17], [16], [15].

Because mass is more fundamental than density, we show in this paper that a density estimator can be constructed from mass. The key advantage of mass is that it can be computed very quickly. The new density estimator based on mass inherits this advantage and executes significantly faster than existing density estimators such as KDE and k-NN. It raises the capability of density-based algorithms to handle large data sets to a new high level.

A mass base function is defined as follows by [16]

$$\mathbf{m}(T(\mathbf{x})) = \begin{cases} \mathsf{m} & \text{if } \mathbf{x} \text{ is in a region of } T(\cdot), \\ 0 & \text{otherwise,} \end{cases}$$

where $T(\cdot)$ is function which subdivides the feature space into non-overlapping regions based on the given data set D; and m is the number of samples in a region of $T(\mathbf{x})$ in which $\mathbf{x}$ falls into.

[16] shows that mass can also be effectively estimated using data subsets $\mathcal{D}_i \subset D$ $(i = 1, \ldots, t)$ and its associated $T_i(\mathbf{x}|\mathcal{D}_i)$, where $|\mathcal{D}_i| = \psi \ll n$. Each $\mathcal{D}_i$ is sampled without replacement from D. The mass estimated using subsamples is defined as

$$\overline{mass}(\mathbf{x}) = \frac{1}{t} \sum_{i=1}^{t} \mathbf{m}(T_i(\mathbf{x}|\mathcal{D}_i)).$$

We now introduce the new density estimators based on mass (DEMass) and describe its implementation in the next two subsections.

A. DEMass

Once mass is estimated, density can be estimated as a ratio of mass and volume.

Thus, the new density estimators based on mass functions $\mathbf{m}(T(\mathbf{x}))$ and $\mathbf{m}(T_i(\mathbf{x}|\mathcal{D}_i))$ are defined respectively as

$$f_{\mathbf{m}}(\mathbf{x}) = \frac{\mathbf{m}(T(\mathbf{x}))}{nv}. \tag{1}$$

$$\bar{f}_{\mathbf{m}}(\mathbf{x}) = \frac{1}{t} \sum_{i=1}^{t} \frac{\mathbf{m}(T_i(\mathbf{x}|\mathcal{D}_i))}{\psi v_i}. \tag{2}$$

where v and v_i are the volumes of regions $T(\mathbf{x})$ and $T_i(\mathbf{x}|\mathcal{D}_i)$, respectively.

We use the term DEMass to refer to density estimator $\bar{f}_{\mathbf{m}}(\mathbf{x})$ in the rest of this paper.

DEMass has two key differences/advantages when compared to the one based on a kernel method or k-NN:

- $\bar{f}_{\mathbf{m}}$ is estimated from $t\psi$ instances only which are significantly smaller than D in a large data set. It sums over t number of randomly generated regions; whereas $\bar{f}_{KDE}$ sums over n number of instances in D, and $\bar{f}_{kNN}$ also requires access to the entire data set. For a large data set, $\bar{f}$ is prohibitively expensive to compute in these two methods [1].
- $\bar{f}_{\mathbf{m}}$ needs no distance measures.

B. Implementation

Mass estimation can be implemented in different ways [17], [16], [15].

When $T(\cdot|\mathcal{D})$ is implemented using a binary tree, the volumes of regions in $T(\cdot|\mathcal{D})$ are controlled by a parameter h which defines the level of binary subdivision.

Let Δ_i be a work space in $\mathcal{R}^d$ which envelops $\mathcal{D}_i$; and Δ_i has its length along each dimension j as $\Delta_{ij} = \max(x_{kj}|\mathbf{x}_k \in \mathcal{D}_i) - \min(x_{kj}|\mathbf{x}_k \in \mathcal{D}_i)$. Each $T_i(\cdot|\mathcal{D}_i)$ is constructed within work space Δ_i, resulting in 2^{hd} hyper-rectangular regions where every region has an equi-width $\delta x_{ij} = \Delta_{ij}/2^h$ on each dimension j and a volume $v_i = \delta x_{i1} \times \cdots \times \delta x_{id}$. For example, in a one-dimensional space with work space Δ_i derived from $\mathcal{D}_i$ and set $h = 3$, $T_i(\cdot|\mathcal{D}_i)$

[1] While there are ways to reduce the computational cost of KDE and k-NN, they are usually limited to low dimensional problems or incur significant preprocessing cost. See Section VII for a discussion.

subdivides the work space into 2^3 equi-width regions. We use T_i to denote T_i^h, unless h is required in the context; and T_i is built from $\mathcal{D}_i$, for each i.

We use the implementation of $T(\cdot|\mathcal{D})$ as described in [16] as the basis to build density estimator $\bar{f}_{\mathbf{m}}$. The algorithm, used to generate such a tree, is given in Appendix A for ease of reference.

The time complexity of constructing the trees is $O(t\psi hd)$. The space complexity is $O(thd + n)$ during construction. After the trees are built, the data set is discarded, yielding $O(thd)$.

To estimate the density of a given instance $\mathbf{x}$, only these trees are used according to Equation (2).

In the next section, we will show that the bias between $\bar{f}_{\mathbf{m}}(\mathbf{x})$ and the true probability density function $p_d(\mathbf{x})$ converges asymptotically.

IV. Error Analysis through Bias-Variance Decomposition

The density estimator based on mass (DEMass) $\bar{f}_{\mathbf{m}}(\mathbf{x})$ can be thought of as a random variable because of its dependence on D and its random subsamples $\mathcal{D}_i$ $(i = 1, \ldots, t)$. Accordingly, we analyze Mean Squared Error (MSE) of $\bar{f}_{\mathbf{m}}(\mathbf{x})$ from its true probability density $p_d(\mathbf{x})$. It is defined as

$$MSE(\bar{f}_{\mathbf{m}}(\mathbf{x})) = E\big[\{\bar{f}_{\mathbf{m}}(\mathbf{x}) - p_d(\mathbf{x})\}^2\big]$$

where the expectation $E[\cdot]$ is taken over the distribution of $\bar{f}_{\mathbf{m}}(\mathbf{x})$. This is rewritten by introducing the expectation of $\bar{f}_{\mathbf{m}}(\mathbf{x})$: $E[\bar{f}_{\mathbf{m}}(\mathbf{x})]$ as follows [13].

$$\begin{aligned} MSE(\bar{f}_{\mathbf{m}}(\mathbf{x})) &= \big\{E[\bar{f}_{\mathbf{m}}(\mathbf{x})] - p_d(\mathbf{x})\big\}^2 \\ &\quad + E\big[\{\bar{f}_{\mathbf{m}}(\mathbf{x}) - E[\bar{f}_{\mathbf{m}}(\mathbf{x})]\}^2\big]. \end{aligned}$$

The first term on the rhs is called "*square bias* " and the second "*variance*." We evaluate the magnitude of each of these two terms in the following.

To simplify notations for the rest of the paper, we have used $T_i(\mathbf{x})$ to denote $T_i(\mathbf{x}|\mathcal{D}_i)$, and $p(T_i(x))$ to denote $p(\mathbf{x}_k \in T_i(\mathbf{x})|\ \mathbf{x}_k \in \mathcal{D}_i)$.

Let $\mathbf{c}_i$ be the center of a region of $T_i(\mathbf{x})$ where each element c_{ij} of $\mathbf{c}_i$ is a middle point of the interval on each dimension j. The second order Taylor approximation of $p_d(\mathbf{x})$ around $\mathbf{c}_i$ for $T_i(\mathbf{x})$ is given as

$$\begin{aligned} p_d(\mathbf{x})|_{\mathbf{c}_i \in T_i(\mathbf{x})} &\approx p_d(\mathbf{c}_i) + (\mathbf{x} - \mathbf{c}_i)^{\mathsf{T}} \nabla p_d(\mathbf{x})|_{\mathbf{x}=\mathbf{c}_i} \\ &\quad + \frac{1}{2}\{(\mathbf{x} - \mathbf{c}_i)^{\mathsf{T}} \nabla\}^2 p_d(\mathbf{x})|_{\mathbf{x}=\mathbf{c}_i}, \quad (3) \end{aligned}$$

where $\nabla = [\partial/\partial x_1, \ldots, \partial/\partial x_d]^{\mathsf{T}}$.

Note that $\mathbf{m}(T_i(\mathbf{x}))$ follows a binomial distribution[2] $B(\psi, p(T_i(\mathbf{x})))$. Therefore, $E[\bar{f}_{\mathbf{m}}(\mathbf{x})]$ is expressed by substituting $E[\mathbf{m}(T_i(\mathbf{x}))] = \psi p(T_i(x))$ in Eq. (2).

$$\begin{aligned} E[\bar{f}_{\mathbf{m}}(\mathbf{x})] &= \frac{1}{t}\sum_{i=1}^{t} \frac{E[\mathbf{m}(T_i(\mathbf{x}))]}{\psi v_i} \\ &= \frac{1}{t}\sum_{i=1}^{t} \frac{p(T_i(x))}{v_i} \\ &= \frac{1}{t}\sum_{i=1}^{t} \frac{1}{v_i} \int_{T_i(\mathbf{x})} p_d(\mathbf{x}_*) d\mathbf{x}_*. \quad (4) \end{aligned}$$

Accordingly, the square bias is evaluated as follows by applying Eq. (3) and the fact that the integral of an odd function over $[c_{ij} - \delta x_j/2, c_{ij} + \delta x_j/2]$ for each dimension j is zero.

$$\begin{aligned} &\big\{E[\bar{f}_{\mathbf{m}}(\mathbf{x})] - p_d(\mathbf{x})\big\}^2 \\ &\approx \Bigg[\frac{1}{t}\sum_{i=1}^{t}\Bigg\{\frac{1}{24}\sum_{j=1}^{d} \frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\Big|_{\mathbf{x}=\mathbf{c}_i} \delta x_{ij}^2 \\ &\quad -(\mathbf{x} - \mathbf{c}_i)^{\mathsf{T}} \nabla p_d(\mathbf{x})|_{\mathbf{x}=\mathbf{c}_i} \\ &\quad -\frac{1}{2}\{(\mathbf{x} - \mathbf{c}_i)^{\mathsf{T}} \nabla\}^2 p_d(\mathbf{x})|_{\mathbf{x}=\mathbf{c}_i}\Bigg\}_{\mathbf{c}_i \in T_i(\mathbf{x})}\Bigg]^2 \\ &\le \Bigg[\frac{1}{t}\sum_{i=1}^{t}\Bigg\{\frac{1}{24}\Bigg|\sum_{j=1}^{d} \frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\Big|_{\mathbf{x}=\mathbf{c}_i}\Bigg| \Delta_{ij}^2 2^{-2h} \\ &\quad + \sum_{j=1}^{d}\Bigg|\frac{\partial p_d(\mathbf{x})}{\partial x_j}\Big|_{\mathbf{x}=\mathbf{c}_i}\Bigg| \Delta_{ij} 2^{-h} \\ &\quad + \frac{1}{2}\sum_{j=1}^{d}\sum_{k=1}^{d}\Bigg|\frac{\partial^2 p_d(\mathbf{x})}{\partial x_j \partial x_k}\Big|_{\mathbf{x}=\mathbf{c}_i}\Bigg| \Delta_{ij}\Delta_{ik} 2^{-2h}\Bigg\}_{\mathbf{c}_i \in T_i(\mathbf{x})}\Bigg]^2 \\ &= O(4^{-h}) \end{aligned}$$

This result shows that the square bias diminishes as level h increases, *i.e.*, as the size of the regions decreases. Though this analysis uses the second order approximation of $p_d(\mathbf{x})$, the result using the higher order approximation is the same since the first order term dominates in the above formula.

Because $\mathbf{m}(T_i(\mathbf{x}))$ follows the binomial distribution $B(\psi, p(T_i(x)))$, the variance of $\mathbf{m}(T_i(\mathbf{x}))$ is

$$var[\mathbf{m}(T_i(\mathbf{x}))] = \psi p(T_i(x))(1 - p(T_i(x))).$$

In concert with Eq. (2), the variance of $\bar{f}_{\mathbf{m}}(\mathbf{x})$ is represented as follows.

$$\begin{aligned} &E\big[\{\bar{f}_{\mathbf{m}}(\mathbf{x}) - E[\bar{f}_{\mathbf{m}}(\mathbf{x})]\}^2\big] \\ &= \frac{1}{t^2}\sum_{i=1}^{t} \frac{p(T_i(x))(1 - p(T_i(x)))}{\psi v_i^2} \\ &= \frac{1}{t^2}\sum_{i=1}^{t} \frac{1}{\psi v_i^2} \int_{T_i(\mathbf{x})} p_d(\mathbf{x}_*) d\mathbf{x}_* \Bigg(1 - \int_{T_i(\mathbf{x})} p_d(\mathbf{x}_*) d\mathbf{x}_*\Bigg). \end{aligned}$$

[2]The implementation of $T(\cdot)$ used in this paper is a tree-based nonparametric method. The binomial distribution is required for the error analysis only.

Using the similar calculus as applied to the square bias, we obtain the variance as follows where $\mathbf{c_i}$ is a center of $T_i(\mathbf{x})$.

$$
\begin{aligned}
&E\big[\{\bar{f}_{\mathbf{m}}(\mathbf{x}) - E[\bar{f}_{\mathbf{m}}(\mathbf{x})]\}^2\big] \\
&\approx \frac{1}{t^2}\sum_{i=1}^{t}\frac{1}{\psi}\left\{p_d(\mathbf{c}_i) + \frac{1}{24}\sum_{j=1}^{d}\frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\Big|_{\mathbf{x}=\mathbf{c}_i}\delta x_{ij}^2\right\} \\
&\quad\times\left\{\frac{1}{v_i} - p_d(\mathbf{c}_i) - \frac{1}{24}\sum_{j=1}^{d}\frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\Big|_{\mathbf{x}=\mathbf{c}_i}\delta x_{ij}^2\right\}. \\
&= \frac{1}{t^2}\sum_{i=1}^{t}\frac{1}{\psi}\left\{p_d(\mathbf{c}_i) + \frac{1}{24}\sum_{j=1}^{d}\frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\Big|_{\mathbf{x}=\mathbf{c}_i}\Delta_{ij}^2 2^{-2h}\right\} \\
&\quad\times\left\{\frac{2^{dh}}{\prod_{j=1}^{d}\Delta_{ij}} - p_d(\mathbf{c}_i) - \frac{1}{24}\sum_{j=1}^{d}\frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\Big|_{\mathbf{x}=\mathbf{c}_i}\Delta_{ij}^2 2^{-2h}\right\}. \\
&\quad= O(2^{dh})
\end{aligned}
$$

This result indicates that the variance increases when level h increases. Also, the result does not change even if we use the higher order approximation because the term $p_d(\mathbf{c}_i)/v_i$ dominates in the above formula.

The property of DEMass, revealed from this error analysis, is similar to that of the conventional kernel density estimator which shows a bias-variance trade off—the bias decreases as the kernel bandwidth b decreases but this increases the variance; and the reverse is true if the kernel bandwidth in increased [13]. The parameter k in k-NN density estimator has the same effect.

In conclusion, DEMass has a comparable performance with the kernel density estimator if both trade-off bias and variance equally well; and it is indeed the case in practice. Figure 1 shows the estimation result of a normal distribution using KDE and DEMass, respectively. It demonstrates that DEMass produces similar result to that generated by KDE, for different data sizes. Smoothing can be applied by increasing b for KDE or decreasing h for DEMass which produces the estimation results as shown in Figure 2. The parameters used for DEMass are: $t = 1000$ and $\psi = n$ when $n = 10, 100$; $\psi = 1000$ when $n = 1000000$.

Note that in either settings shown in Figures 1 and 2, the estimations of both KDE and DEMass approach the true distribution as the number of instances increases.

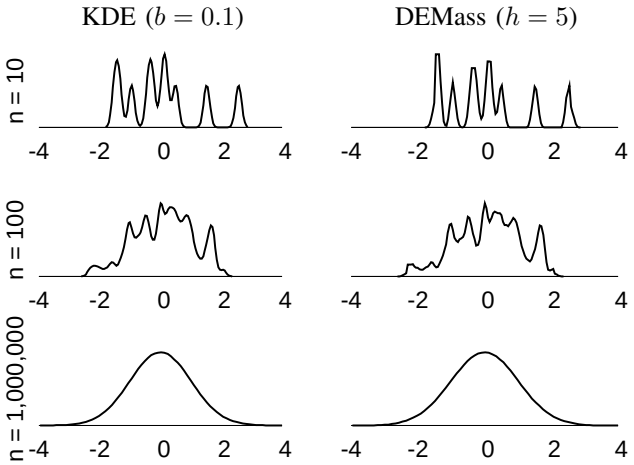


Figure 1: Example estimations of Kernel Density Estimator (with Gaussian kernel) using $b = 0.1$ and DEMass using $h = 5$ for different data sizes, $n = 10, 100, 1000000$. The true data distribution is a normal distribution.

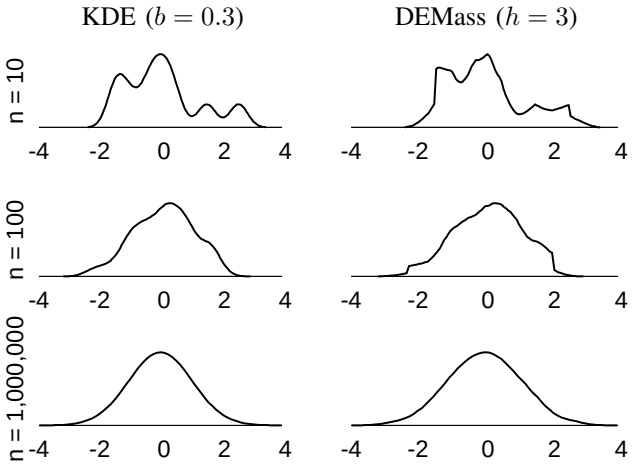


Figure 2: Example estimations of Kernel Density Estimator (with Gaussian kernel) using $b = 0.3$ and DEMass using $h = 3$ for the same data used in Figure 1.

V. Using DEMass in existing density-based algorithms

This section describes how DEMass can be applied to two current density-based algorithms, DBSCAN [6] and LOF [4], in place of their existing density estimators. DBSCAN and LOF are one of the best algorithms for clustering and anomaly detection, respectively.

Using DEMass in both DBSCAN and LOF automatically carries the two advantages mentioned in Section III: (i) The estimation requires no distance measures, thus, it completely saves the cost of distance calculations for every pair of instances; and (ii) DEMass enables small samples to construct the required regions $T(\cdot|\mathcal{D})$, overcoming the key limitation of DBSCAN and LOF in handling very large databases. We will discuss further advantages specific to individual algorithms in the following subsections.

A. DEMass-DBSCAN

The principal steps of DEMass-DBSCAN is the same as DBSCAN, except that no border points and their associated step are required. A comparison of the two algorithms are provided in Table I. The algorithm for DBSCAN is adapted from [14].

While following its principal steps, the use of DEMass simplifies DBSCAN in two ways, in addition to the two

step	DBSCAN	DEMass-DBSCAN
1	Label all points as core, border, or noise points, based on $\bar{f}_{kNN}(\mathbf{x})$	Label all $T(\mathbf{x})$ satisfying Definition 1 as core regions, based on $\bar{f}_{\mathbf{m}}(\mathbf{x})$. Points not covered by core regions are noise.
2	Eliminate noise points	Eliminate noise points
3	Connect all core points that are within ϵ of each other.	Connect all core regions that have non-zero intersections.
4	Make each group of connected core points into a separate cluster	Make each group of connected core regions into a separate cluster.
5	Assign each border point to one of the clusters of its associated core points.	

Table I: Algorithms for DBSCAN and DEMass-DBSCAN. Note that border points are not required with DEMass-DBSCAN; thus step 5 is not needed. Both versions of DBSCAN could include an additional cluster size threshold to eliminate small size clusters in the last step.

advantages already mentioned above. First, DEMass enables regions to be labelled instead of individual points. Because the number of regions is significantly less than the number of points, linking regions to form a cluster becomes significantly faster than connecting points. Second, no border points need to be defined because the connections within a cluster are established via core regions only when DEMass is used. The first simplification is the key reason for the significant speed up achieved by DEMass-DBSCAN, which we will show in Section VI-A.

The formal definitions for DEMass-DBSCAN are given as follows. Although point-based definitions can be similarly defined as in DBSCAN [6], we show that set-based definitions are simpler.

Definition 1: $T(\mathbf{x})$ is a **core region** of point $\mathbf{x}$ wrt h and $MinPts$ if $\mathbf{m}(T(\mathbf{x})) \geq MinPts$.

Definition 2: $T_r(\cdot)$ is **density-connected** to $T_s(\cdot)$ wrt h and $MinPts$ if there is a chain of regions $T_1(\cdot), \ldots, T_g(\cdot)$ where $r = 1$ and $s = g$ such that $T_i(\cdot) \cap T_{i+1}(\cdot) \neq \emptyset$ and $T_i(\cdot)$ is a core region for all i wrt h and $MinPts$.

Definition 3: An **arbitrary-shape cluster** C wrt h and $MinPts$ is a non-empty subset of a database D satisfying the following condition: $\forall r, s; T_r(\cdot), T_s(\cdot) \subset C$: $T_r(\cdot)$ is density-connected to $T_s(\cdot)$ wrt h and $MinPts$.

Definition 4: Let $C_1, \ldots, C_k$ be the clusters of D wrt h and $MinPts$. **Noise** is the set of points in D not belonging to any cluster C_j, i.e., noise = $\{\mathbf{x} \in D | \forall j : \mathbf{x} \notin C_j\}$.

Definitions 1 and 2 assume that $T_i(\cdot)$ has the same volume for every i. In the case $T_i(\cdot)$ has a different volume for each i (as in our implementation described in Section III-B), only the condition in Definition 1 needs to be modified to $\mathbf{m}(T(\mathbf{x}))\frac{v_{max}}{v} \geq MinPts$, where $v_{max} = \max_i v_i$, and v is the volume of region $T(\mathbf{x})$. The normalisation factor $\frac{v}{v_{max}}$ ranges from 0 to 1.

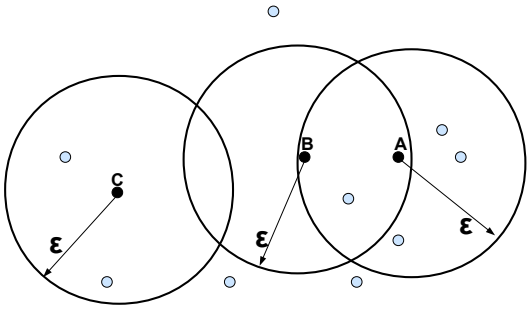


Figure 3: An example for DBSCAN for $Minpts = 5$. A is a core point, B is a border point, and C is a noise point.

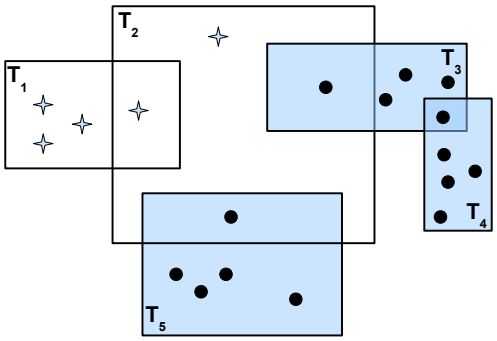


Figure 4: An example for DEMass-DBSCAN for $Minpts = 5$. The circle symbol indicates core points and the star symbol indicates noise points. T_3, T_4, and T_5 are core regions. T_3 and T_4 are linked by a common core point.

A comparison between DBSCAN and DEMass-DBSCAN is provided using two examples showed in Figures 3 and 4. They show how core points and non-core points are labelled in DBSCAN and DEMass-DBSCAN. Note that hyper-spheres are used in DBSCAN, and hyper-rectangles are used in DEMass-DBSCAN in our current implementation of $T_i(\cdot)$.

B. DEMass-LOF

Table II compares the algorithms for LOF and DEMass-LOF which have three identical principal steps: Compute density distribution and LOF, and then rank all instances based on their LOF values. The key difference is the density estimator used in step 1 which changes the computation of LOF in step 2.

In addition to the two advantages due to the use of DEMass mentioned in Section III, the advantage specific to LOF is that DEMass enables the computation of the relative density to be substantially simplified, changing from nearest-neighbour-based to set-based. Instead of finding the neighbours of $\mathbf{x}$ and then compute the density of each neighbours, the modified ranking measure LOF_p is computed based on the region $T(\mathbf{x})$ and its immediate larger region $\breve{T}_i(\mathbf{x}) \supset T_i(\mathbf{x})$. In a tree implementation of $T(\cdot)$,

step	LOF	DEMass-LOF
1	Compute density distribution: $\bar{f}_{kNN}(\mathbf{x})$	Compute density distribution: $\bar{f}_{\mathbf{m}}(\mathbf{x})$
2	Compute $LOF(\mathbf{x})$ using $\frac{\sum_{\mathbf{x}' \in N(\mathbf{x},k)} \frac{\bar{f}_{kNN}(\mathbf{x}')}{\lvert N(\mathbf{x},k)\rvert}}{\bar{f}_{kNN}(\mathbf{x})}$	Compute $LOF_p(\mathbf{x})$ using: $\frac{\frac{1}{t}\sum_{i=1}^{t} \frac{\mathbf{m}(\breve{T}_i(\mathbf{x}))}{\breve{\psi}\breve{v}_i}}{\bar{f}_{\mathbf{m}}(\mathbf{x})}$
3	Rank all instances based on their LOF values in descending order	Rank all instances based on their LOF_p values in descending order

Table II: Algorithms for LOF and DEMass-LOF. $\bar{f}_{kNN}(\mathbf{x})$ and $N(\mathbf{x}, k)$ are defined in Section II-B; $\bar{f}_{\mathbf{m}}(\mathbf{x})$ is defined in Section III. $\breve{T}_i(\mathbf{x}) \supset T_i(\mathbf{x})$ correspond to the parent and child nodes in our tree implementation; $\breve{\psi}$ and $\breve{v}_i$ are the data size and volume of $\breve{T}_i(\mathbf{x})$, respectively. Note that $\breve{T}_i(\mathbf{x})$, the next superset of $T_i^h(\mathbf{x})$, is not necessarily $T_i^{h-1}(\mathbf{x})$ because there are d levels in the tree for each increment of h and the implementation allows single branch extensions if there are no data in other branches. See Appendix A for the details of the implementation.

this corresponds to computing the density of the node in which x falls into, relative to the density of its parent node.

In steps 1 and 2, the time complexities of DEMass-LOF and LOF are $O(nt\psi)$ and $O(n^2)$, respectively. Since DEMass-LOF does not need to perform neighbourhood search as in LOF, it is much faster, especially in large data sets. This is because DEMass-LOF does not need to compute the density of all neighbours of each instance.

The parameter k in LOF has an inverse relationship with h in DEMass-LOF, i.e., high h corresponds to low k (which covers a smaller region than that using low h or high k).

A larger k increases LOF's processing time so as a larger h increases DEMass-LOF's processing time.

Note that LOF is a relative density score. Both LOF and LOF_p range from 0 to $+\infty$, indicating the degree of anomaly; the higher the LOF score, the higher the degree of anomaly.

VI. Empirical evaluation

Both of the following evaluations, in clustering and anomaly detection tasks, are conducted in the unsupervised learning setting. We will compare DBSCAN with DEMass-DBSCAN in the first subsection and then compare LOF with DEMass-LOF in the second subsection.

All experiments were conducted as single thread jobs processed at 2.3 GHz in a Linux cluster (www.vpac.org) using a node with 32 GB memory. All DEMass related algorithms were written in JAVA in WEKA platform [19], so as DBSCAN. LOF was written in Java in ELKI platform version 0.4 [1]. The data sets used are from UCI Machine Learning Repository [7], unless stated otherwise.

The clustering result was reported in terms of CPU runtime (in seconds), number of clusters identified, number of unassigned instances, and F-measure which was calculated based on assigned instances only. F-measure = 1 when all assigned instances are in the correct clusters, i.e., perfect clustering; and F-measure = 0 if all instances are assigned to wrong clusters. The anomaly detection result was reported in terms of CPU runtime and AUC (Area Under ROC Curve) based on the ranked result. We tuned the parameters of each algorithm and reported the best result.

Table III: Clustering results in the Ring-Curve+Wave+Tri-Gaussian data sets for DEMass-DBSCAN ($h = 7$ for 3-dimensional data; $h = 6$ for 48-dimensional data) and DBSCAN ($\epsilon = 0.01$).

	3-dimensional data		48-dimensional data	
	DEMass-DBSCAN	DBSCAN	DEMass-DBSCAN	DBSCAN
Runtime	135	2391	1261	21906
#cluster [7]	9	8	7	8
#unassigned	535	332	61	332
F-measure	0.9999	0.9999	1.0000	0.9999

A. DEMass-DBSCAN versus DBSCAN

DEMass-DBSCAN had $\psi = 256$ and $t = 1000$ as default; and both DEMass-DBSCAN and DBSCAN used $MinPts = 6$ in all experiments. As a result, only one parameter needed to be tuned for a particular data set: h for DEMass-DBSCAN and ϵ for DBSCAN.

Ring-Curve-Wave-Tri-Gaussian. It has three two-dimensional synthetic data embedded in either a 3-dimensional data set or a 48-dimensional data set (where 42 dimensions are irrelevant with a constant value). The three two-dimensional data are Ring-Curve, Wave and Triangular-Gaussian shown in Figure 7 in Appendix B, which have a total of seven clusters. Each cluster has 10,000 instances with a total of 70,000 instances.

The clustering results from DEMass-DBSCAN and DBSCAN are shown in Table III. DEMass-DBSCAN ran faster than DBSCAN by a factor more than 17 in both data sets. In terms of #clusters and #unassigned, DEMass-DBSCAN performed slightly worse than DBSCAN in the 3-dimensional data set, but better in the 48-dimensional data set. DEMass-DBSCAN decreased its number of unassigned instances from 535 to 61 when the number of dimensions was increased from 3 to 48; whereas DBSCAN had the same 332 unassigned instances in both cases. DEMass-DBSCAN performs either similarly to or better than DBSCAN in terms of F-measure in these two data sets.

In order to examine how well the algorithms scale up to large data size, we used the 48-dimensional data set and increased the data size from 7000 to 70000, half-a-million, 1 million and 10 million. Figure 5 plotted runtime ratio versus data size ratio (1, 10, 75, 150 and 1500) by using 7000 as the base. The result showed that DEMass-DBSCAN had a sublinear increase in runtime: The runtime ratio increased from 1 to 101 when the data size ratio increased from 1 to

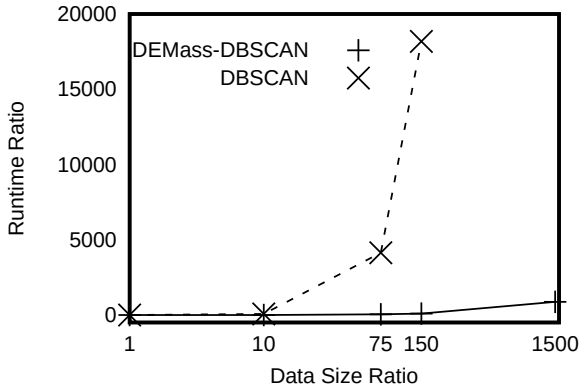


Figure 5: Scale up: DEMass-DBSCAN vs DBSCAN in the 48-dimensional Ring-Curve-Wave-TriGaussian data set. Note that DBSCAN completed the task of the one-million data set (at data size ratio=150) in 36 days versus DEMass-DBSCAN's 4.5 hours. Even with the 10-million data set, DEMass-DBSCAN completed it in 38 hours.

150. In contrast, DBSCAN's runtime ratio increased from 1 to 18000 with the same increase in data size ratio. DEMass-DBSCAN was faster than DBSCAN by a factor of 193 when the one-million data set is used. Even the data size was increased by a factor of 1500, the runtime of DEMass-DBSCAN increased by a factor of 862 only.

OneBig and Pendigits. The OneBig data set [11] has 20 attributes, 9 clusters and a total of 68,000 instances. The biggest cluster has 50,011 instances, and each of the other eight clusters has approximately 1000 instances. In addition, there are 10,000 noise instances randomly distributed in the feature space. The Pendigits data set has 16 attributes and 10 clusters. Each cluster has approximately 1,100 instances which makes up a total of 10,992 instances.

The result in Table IV showed that DEMass-DBSCAN and DBSCAN for OneBig had the same clustering result in terms of F-measure and number of clusters; but DEMass-DBSCAN ran faster than DBSCAN by a factor of 7. Note that DEMass-DBSCAN had correctly identified all but one of the 10,000 noise instances; whereas DBSCAN correctly identified all of the noise instances. For Pendigits, the result showed that although DEMass-DBSCAN had a lower F-Measure than DBSCAN, it was better than DBSCAN in all other measures: it had only 20% instances unassigned whereas DBSCAN had 57% instances unassigned; DEMass-DBSCAN found 47 cluster whereas DBSCAN detected 65.

B. DEMass-LOF versus LOF

For anomaly detection tasks, we compared LOF with DEMass-LOF in this section. Table V provided the properties of the data sets used. Note that Http and Smtp are subsets of the network intrusion data set used in KDDCUP 99 [20]; and an anomaly data generator [12] is used to generate a synthetic data set. All the data sets used have

Table IV: Clustering results in the OneBig and Pendigits data sets for DEMass-DBSCAN ($h = 3$ for OneBig; $h = 2$ for Pendigits) and DBSCAN ($\epsilon = 0.1$ for OneBig; $\epsilon = 0.2$ for Pendigits).

	OneBig		Pendigits	
	DEMass-DBSCAN	DBSCAN	DEMass-DBSCAN	DBSCAN
Runtime	1145	8544	91	204
#cluster	[9] 9	9	[10] 47	65
#unassigned	10021	10005	2166	6251
F-measure	1.00	1.00	0.65	0.75

Table V: Data sets used for the anomaly detection task for comparing DEMass-LOF with LOF.

Data	Size n	d	anomaly class
Http	567,497	3	attack (0.4%)
ForestCover (FC)	286,048	10	class 4 (0.9%) vs. class 2
Mulcross	262,144	4	2 clusters (10%)
Smtp	95,156	3	attack (0.03%)
Shuttle	49,097	9	classes 2,3,5,6,7 (7%)

nearly fifty thousand or more instances, with the largest up to half a million instances. The default settings for DEMass-LOF were $\psi = 256$ and $t = 100$.

Table VI compares LOF with DEMass-LOF in terms of detection performance AUC and time. DEMass-LOF using either h=1 or 4 obtained better AUC results than LOF. It is interesting to note that DEMass-LOF achieved extreme results in the Smtp and Mulcross data sets between the two h settings; and it behaved differently in these two data sets, where a low h setting is better in Mulcross but a high h setting is better in Smtp. This is because the two data sets have two different types of anomalies: clustered and scattered anomalies [10]. Mulcross has clustered anomalies, i.e., outlying clusters with high density but a small number of instances. DEMass-LOF with a high h setting (i.e., h=4) regarded these anomaly clusters more 'normal' than normal instances, which was reflected in the result: AUC=0.09. In contrast, the Smtp data set has scattered anomalies which

Table VI: Compare LOF and DEMass-LOF in terms of AUC (Area Under ROC Curve) and time (in seconds). AUC=1 is the perfect detection performance and AUC=0 is the worst. The default settings for DEMass-LOF were $h = 1$, $\psi = 256$ and $t = 100$ which were used for all data sets. The parameter k (for LOF) and h (for DEMass-LOF) were changed in order to explore a better result.

	AUC				Time (seconds)			
	LOF		DEMass-LOF		LOF		DEMass-LOF	
	k=10	k=60	h=1	h=4	k=10	k=60	h=1	h=4
Http	0.44	0.35	0.99	0.93	18913	19818	19	42
FC	0.57	0.58	0.74	0.77	10835	11147	39	40
Mulcross	0.59	0.59	0.96	0.09	5432	5486	12	53
Smtp	0.32	0.85	0.29	0.89	540	552	2	5
Shuttle	0.55	0.62	0.94	0.71	368	380	5	12

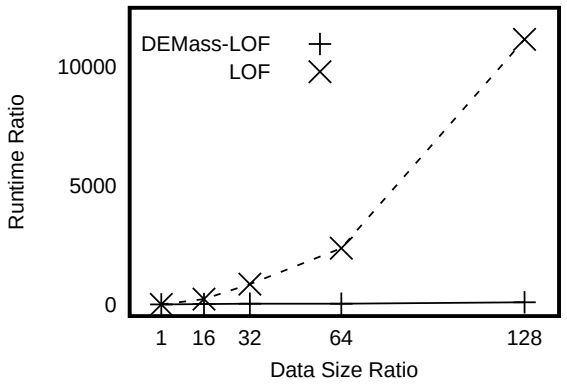


Figure 6: Scale up: LOF versus DEMass-LOF in Mulcross. The base for data size ratio is 8192 instances and the base for runtime ratio is the runtime on 8192 instances.

are isolated outlying instances around normal clusters. This scenario requires a high h setting in order for DEMass-LOF to compute the right densities for these anomalies.

LOF was not competitive, and the AUC results did not change much from the presented results even other k values were used (we had tried k=30, 40, 50, 80, 100, 120.)

However, it shall be noted that LOF could achieve good detection accuracy with an appropriate k. For example, LOF obtained AUC=0.99 when $k = 4000$ was used in the shuttle data set. But similar search in the largest three data sets failed with out of memory problem even though the computer system was allocated 32 GB memory! This result reveals two universal problems with k-NN approaches like LOF: (i) An extensive parameter search is required to obtain good detection accuracy; this search adds a significant cost to the already long runtime process. The total time cost is often prohibitive; and (ii) high memory requirement.

Table VI also compares these detectors in terms of processing time. DEMass-LOF was one to near-three orders of magnitude faster than LOF in these data sets.

Figure 6 showed the runtime of both algorithms when scaling from 8192 instances up to a million instances in the Mulcross data set. The data size was increased by a factor of 16, 32, 64, 128 from 8192 instances. DEMass-LOF increased its runtime by a factor of 11, 23, 25 and 86, respectively. In contrast, LOF increased its runtime by a factor of 217, 845, 2371, 11173, respectively. At data size ratio = 128, which has a million instances, LOF completed the task in 28 hours whereas DEMass-LOF accomplished it in 45 seconds!

VII. Discussion

What we have presented is the first density estimation method that utilizes no distance measures. It potentially solves fundamental problems such as the curse of dimensionality in which the use of a distance measure plays a key part in creating the problem [2], [8].

There are significant improvements of nearest neighbour search in recent times. For example, indexing schemes to speed up nearest neighbour search such as Cover Trees [3] and M-Trees [5] are claimed to have time complexity significantly better than $O(n^2)$. Indexing schemes such as Cover Trees or M-Trees rely on distance-based pruning methods in both the index tree construction and range query processes. Distance-based pruning methods cannot scale up to massive data, and they are known to be inefficient even for a moderate number of dimensions. Thus, it is unlikely that any of the recent indexing schemes can be used to speed up nearest neighbour search to the level that has been achieved already by DEMass-DBSCAN and DEMass-LOF, especially in large data sets.

Note that the purpose of trees used in DEMass differs from that used for Cover Trees or M-Trees. Trees in DEMass are used to estimate mass, the core computation process. In contrast, Cover Trees or M-Trees are indices used to speed up nearest neighbour search. The indices are required because the core computation, i.e., the requirement to calculate distance for every pair of instances, is slow. In other words, one uses trees directly in the core process; and the other uses trees to aid the core process where trees are not used in the actual computation of distance.

The cost of KDE estimation can be lowered, for example, by reducing the given data set D to some 'representative' subset, where each representative kernel is derived from a subsample using a maximum likelihood method such as EM. This reduces the KDE estimation time; but it comes with a cost of an expensive pre-processing step.

It is possible to use neighbours to compute LOF for DEMass-LOF. However, the runtime advantage over LOF will be significantly reduced because of the additional computations required to calculate the density of each neighbour, even though it does not need to find neighbours based on distance calculations.

DENCLUE [9], a generic density-based algorithm, builds a density distribution from data, and then uses a threshold to determine clusters—all connected points above the threshold form a cluster. DBSCAN is a special case of DENCLUE. DEMass-DENCLUE has exactly the same procedure as DEMass-DBSCAN, where $Minpts$ or the equivalent density threshold stated in Section V-A is employed as the threshold.

DEMass sets a new benchmark of what density-based algorithms can achieve. In contrast to the density-based approaches, mass-based approaches [17], [16] solve problems without the use of a density estimator. Mass-based approaches have been shown to perform better than the current density-based approaches in terms of time and space complexities. It is thus interesting to compare the new benchmark achieved by DEMass-density-based approaches with mass-based approaches.

The current implementation of DEMass has two limita-

tions. First, it has step subdivisions controlled by a global parameter h. The limited possible steps may be too coarse for some applications and the setting is not adaptive to local variations in density. Second, the grid-based implementation carries all the limitations associated with grid-based approaches, especially dealing high dimensional problems. All these limitations can be overcome by using a non-grid method which is adaptive to the local data distribution. This non-grid-based implementation will eliminate one global parameter and potentially tackle high-dimensional problems more effectively.

VIII. Conclusions and Future Work

The new density estimation method we introduced have two unique features which can not be found in existing density estimation methods. First, it is the first density estimator that utilizes no distance measures. Second, it has average case sublinear time complexity and constant space complexity. Existing density estimators must use a distance measure and have time and space complexities a lot worse than linear. The time and space complexities achieved set a new benchmark for density-based algorithms, of what previously thought impossible.

The bias-variance analysis reveals that the new density estimator has the same characteristic as kernel density estimator, i.e., both have a smoothing parameter used to trade-off between systematic error (bias) and random error (variance).

Making full use of the features in the new density estimator, we show that two current algorithms can be significantly simplified through set-based definitions rather than the current point-based definitions. This has directly contributed to their improved time complexities.

Our evaluation shows that the new density estimator not only successfully replaces existing density estimators in two density-based algorithms, DBSCAN and LOF, but significantly improves their runtime. In addition, DEMass-DBSCAN and DEMass-LOF often achieve equivalent or better task-specific performances than DBSCAN and LOF.

Our result implies that most, if not all, density-based algorithms can reap the immediate benefit of significantly lowering their time complexities by simply replacing the existing density estimators with the new one, with a potential further improvement in the task-specific performance.

Future work has three directions. First, we will apply the new density estimator in existing algorithms in more areas. We will ascertain whether there are areas in which the new density estimator cannot replace existing density estimators. Second, compare DEMass-density-based approaches with mass-based approaches to determine their relative strengths and weaknesses. Third, we will explore DEMass's ability to deal with high dimensional problems.

IX. Acknowledgments

This work is partially supported by the Air Force Research Laboratory, under agreement# FA2386-10-1-4052. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Xiao Yu Ge assisted in experiments using ELKI. Hiroshi Motoda, Zhouyu Fu, and the anonymous reviewers had provided many helpful comments to improve this paper.

The source code of DEMass-DBSCAN is available at http://sourceforge.net/projects/mass-estimation/.

References

[1] E. Achtert, H.-P. Kriegel, and A. Zimek ELKI: A Software System for Evaluation of Subspace Clustering Algorithms In *Proceedings of the 20th International Conference on Scientific and Statistical Database Management*, pages 580-585, 2008.

[2] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory*, pages 217–235, 1999.

[3] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 97–104, 2006.

[4] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying density-based local outliers. In *Proceedings of ACM SIGMOD*, pages 93–104, 2000.

[5] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 426–435, 1997.

[6] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of KDD*, pages 226–231, 1996.

[7] A. Frank and A. Asuncion. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. 2010.

[8] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What is the nearest neighbor in high dimensional spaces? In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 506–515, 2000.

[9] A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proceedings of KDD*, pages 58–65. AAAI Press, 1998.

[10] F. T. Liu, K. M. Ting, and Z.-H. Zhou. On detecting clustered anomalies using SCiForest. In *Proceedings of ECML PKDD*, pages 274–290, 2010.

[11] A. Nanopoulos, Y. Theodoridis, and Y. Manolopoulos. Indexed-based density biased sampling for clustering applications. *IEEE Transaction on Data and Knowledge Engineering*, 57(1):37–63, 2006.

[12] D. M. Rocke, and D. L. Woodruff. Identification of outliers in multivariate data. *Journal of the American Statistical Association*. 91(435):1047–1061. 1996.

[13] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC, 1986.

[14] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2006.

[15] S. C. Tan, K. M. Ting, and F. T. Liu. Fast anomaly detection for streaming data. In *Proceedings of IJCAI. pages 1151–1156*, 2011.

[16] K. M. Ting and J. R. Wells. Multi-dimensional mass estimation and mass-based clustering. In *Proceedings of IEEE ICDM*, pages 511–520, 2010.

[17] K. M. Ting, G.-T. Zhou, F. T. Liu, and S. C. Tan. Mass estimation and its applications. In *Proceedings of ACM SIGKDD*, pages 989–998, 2010.

[18] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Second Edition. Springer, 2000.

[19] I. H. Witten, E. Frank and M.A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Third Edition. Morgan Kaufmann, 2011.

[20] K. Yamanishi, J.-I. Takeuchi, G. Williams, and P. Milne. Online unsupervised outlier detection using finite mixtures with discounting learning algorithms. In *Proceedings of ACM SIGKDD*, pages 320–324, 2000.

APPENDIX A: ALGORITHMS FOR GENERATING BINARY TREES TO REPRESENT $T(\mathbf{x}|\mathcal{D})$

We use the same algorithms to generate binary trees to represent $T(\mathbf{x}|\mathcal{D})$ as used in [16] for multi-dimensional mass estimation. Only the pertinent details are provided here.

Algorithm 1 : BuildTrees(D, t, ψ, h)

Inputs: D - input data, t - number of trees, ψ - sub-sampling size, h - number of times an attribute is employed in a path.
Output: F - a set of t h:d-Trees

$MaxHeightLimit \leftarrow h \times d$
Initialize F
for $i = 1$ to t **do**
  $\mathcal{D} \leftarrow sample(D, \psi)$ {strictly without replacement}
  $(min, max) \leftarrow$ InitialiseWorkSpace($\mathcal{D}$)
  $F \leftarrow F \cup$ SingleTree$(\mathcal{D}, min, max, 0)$
end for

A work space which envelops $\mathcal{D}$ is partitioned into 2^ℓ equal-size regions at the leaves of the tree with height $\ell = h \times d$, where d is the number of dimensions. Let m_k be the mass of region k; and there is a total of 2^ℓ regions which have a total mass: $|\mathcal{D}| = \sum_{k=1}^{2^\ell} \mathsf{m}_k$, where $\mathsf{m}_k = \mathbf{m}(T(\mathbf{x}|\mathcal{D}))$; and $\mathbf{x}$ is in region k of T.

$T(\mathbf{x}|\mathcal{D})$ is represented as a binary tree (called h:d-Tree in [16]), where each path from the root to a leaf has $h \times d$ nodes such that each of the d attributes appears exactly h times.

Algorithm 1 generates t trees from a given data set D. Algorithm 2 generates a single tree using a subset $\mathcal{D} \subset D$, where $|\mathcal{D}| = \psi$.

Algorithm 2 : SingleTree$(\mathcal{D}, min, max, \ell)$

Inputs: $\mathcal{D}$ - input data, min & max - arrays of minimum and maximum values for each attribute in A that define a work space, ℓ - current height level, A - set of d attributes.
Output: an h:d-Tree

while $(\ell < MaxHeightLimit)$ do
{Retrieve an attribute from A based on height level.}
$q \leftarrow nextAttribute(A, \ell)$
$p \leftarrow (max_q + min_q)/2$
$\mathcal{D}_l \leftarrow filter(\mathcal{D}, q < p)$
$\mathcal{D}_r \leftarrow filter(\mathcal{D}, q \geq p)$
if $(|\mathcal{D}_l| = 0)$ or $(|\mathcal{D}_r| = 0)$ **then**
  {Reduce range for single-branch node.}
  if $(|\mathcal{D}_l| > 0)$ **then** $max_q \leftarrow p$
  else $min_q \leftarrow p$
  end if
  $\ell \leftarrow \ell + 1$
  continue at the start of while loop
end if
{Build two nodes: $Left$ and $Right$ as a result of a split into two equal-volume half-spaces.}
$temp \leftarrow max_q$; $max_q \leftarrow p$
$Left \leftarrow$ SingleTree$(\mathcal{D}_l, min, max, \ell + 1)$
$max_q \leftarrow temp$; $min_q \leftarrow p$
$Right \leftarrow$ SingleTree$(\mathcal{D}_r, min, max, \ell + 1)$
endwhile
return $Node(Left, Right, SplitAtt \leftarrow q, SplitValue \leftarrow p, Size \leftarrow |\mathcal{D}|)$

APPENDIX B - DATA CHARACTERISTIC

The characteristic of the data set, Ring-Curve-Wave-TriGaussian, used in Section V-A is shown in Figure 7. Each of the Ring-Curve, Wave and Triangular-Gaussian is a two-dimensional data set; and together there is a total of seven clusters. Each cluster has 10000 instances. When used in the scale up experiment, the data size in each cluster was scaled by a factor of 0.1, 1, 75, 150 to 1500.

(a) Ring-Curve (b) Wave (c) Tri-Gaussian

Figure 7: Scatter plot of the clusters in the Ring-Curve-Wave-TriGaussian data set, as used in [16].

Fast Anomaly Detection for Streaming Data*

Swee Chuan Tan
SIM University, Singapore
jamestansc@unisim.edu.sg

Kai Ming Ting and **Tony Fei Liu**
Monash University, Australia
{kaiming.ting, tony.liu}@monash.edu

Abstract

This paper introduces Streaming Half-Space-Trees (HS-Trees), a fast one-class anomaly detector for evolving data streams. It requires only normal data for training and works well when anomalous data are rare. The model features an ensemble of *random HS-Trees*, and the tree structure is constructed without any data. This makes the method highly efficient because it requires no model restructuring when adapting to evolving data streams. Our analysis shows that Streaming HS-Trees has *constant* amortised time complexity and constant memory requirement. When compared with a state-of-the-art method, our method performs favourably in terms of detection accuracy and runtime performance. Our experimental results also show that the detection performance of Streaming HS-Trees is not sensitive to its parameter settings.

1 Introduction

The problem of detecting anomalies in streaming data has the following characteristics. Firstly, the stream is infinite, so any off-line learning algorithms that attempt to store the entire stream for analysis will run out of memory space. Secondly, the stream contains mostly normal instances because anomalous data are rare and may not be available for training. In this case, any multi-class classifiers that require fully labeled data will not be suitable. Thirdly, streaming data often evolve over time. Thus, the model must adapt to different parts of the stream in order to maintain high detection accuracy.

This paper proposes an anomaly detection algorithm, Streaming Half-Space Trees (HS-Trees), that addresses the above-mentioned problem. The proposed method has several features that distinguish itself from other existing techniques. Firstly, it processes data in one pass and only requires constant amount of memory to process potentially endless streaming data or massive datasets. Thus it is different from existing off-line anomaly detectors (e.g., ORCA [Bay and Schwabacher, 2003], LOF [Breunig *et al.*, 2000] and SVM [Scholkopf *et al.*, 2002]) which are designed to mine static and finite datasets.

Secondly, Streaming HS-Trees is a one-class anomaly detector which is useful when a stream contains a significant amount of normal data.

Thirdly, it performs fast model updates in order to maintain high detection accuracy when dealing with time-varying data distribution. Its model update is simple and fast because it requires no modifications of the tree structure when processing streaming data.

Streaming HS-Trees employs *mass* [Ting *et al.*, 2010] as a measure to rank anomalies. The mass profile can be constructed with small samples, allowing the anomaly detector to learn quickly and adapt to changes in data streams in a timely manner.

Unlike other decision trees (e.g., random forests [Breiman, 2001]), Streaming HS-Trees does not induce its tree structure from actual training examples. Instead, the tree structure is constructed using the data space dimensions alone. The trees can be built quickly because it requires no attribute or split-point evaluations; and the model can be deployed before the streaming data arrive. A direct consequence of this feature is that Streaming HS-Trees has a *constant* amortised time complexity and a constant memory requirement. This is unlike algorithms that induce decision tree and alter its tree structure dynamically as streaming data arrive (e.g., Hoeffding Tree [Domingos and Hulten, 2000]).

Our experimental study shows that an ensemble of Streaming HS-Trees leads to a robust and accurate anomaly detector that is not too sensitive to different parameter settings.

2 Related work

In the literature, there are already a number of studies devoted to anomaly detection in *static* datasets. Typical examples include the statistical methods [Barnett and Lewis, 1994], classification-based methods [Abe *et al.*, 2006], clustering-based methods [He *et al.*, 2003], distance-based methods [Bay and Schwabacher, 2003], One-Class Support Vector Machine (SVM) [Scholkopf *et al.*, 2002] and Isolation Forest [Liu *et al.*, 2008]. These off-line learning methods are not designed to process streaming data because they require loading of the entire dataset into the main memory for mining.

*This work is partially supported by the Air Force Research Laboratory, under agreement# FA2386-10-1-4052. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

Recent work on anomaly detection for streaming data include the domain of monitoring sensor networks [Subramaniam *et al.*, 2006] and for abnormal event detection [Davy *et al.*, 2005], but there is currently little work considering anomaly detection in *evolving* data streams.

One interesting related work is LOADED by Otey *et al.* [2006], a link-based unsupervised anomaly detector that works well on datasets with mixed (continuous and categorical) attributes. However, LOADED does not work well on datasets with purely continuous attributes [Otey *et al.*, 2006]. Unlike LOADED, Streaming HS-Trees is a semi-supervised one-class learner [Chandola *et al.*, 2009] that works well for data with continuous attributes.

A recent system that deals with non-stationary data distributions is OLINDDA (OnLIne Novelty and Drift Detection Algorithm) [Spinosa *et al.*, 2009]. OLINDDA uses standard clustering algorithm to groups examples into clusters (or concepts). Through monitoring the clusters, it detects new emerging concepts rather than anomalies.

Apart from unsupervised methods discussed earlier, supervised learning methods can also be used for anomaly detection in data streams. For example, Hoeffding Trees (HT) [Domingos and Hulten, 2000; Hulten *et al.*, 2001] is an incremental anytime decision tree induction algorithm for classifying high-speed data streams. HT can also be used with Online Coordinate Boosting (denoted as BoostHT) [Pelossof *et al.*, 2009]. HT and BoostHT require positive as well as negative class labels to be available for training. However, this is not a realistic assumption because anomalous data is usually rare or not available for training.

3 The Proposed Method

This section presents the proposed method and the key notations used to describe the method are listed in Table 1.

x	a streaming point
n	the number of streaming points
T	an Half Space Tree, HS-tree
$Node$	a node in an HS-Tree
k	the current depth of a node or $Node.k$
t	the number of HS-Trees in an ensemble
h	maximum depth (level) of a tree, or $maxDepth$
r	mass of a node in the reference window
l	mass of a node in the latest window
ψ	window size
s	an anomaly score

Table 1: Key notations used in this paper.

3.1 Overview

The proposed method is an ensemble of HS-Trees. Each HS-Tree consists of a set of nodes, where each node captures the number of data items (a.k.a. mass) within a particular subspace of the data stream. Mass is used to profile the degree of anomaly because it is simple and fast to compute in comparison to distance-based or density-based methods.

To facilitate learning of mass profiles in evolving data streams, the algorithm segments the stream into windows of equal size (where each window contains a fixed number of data items). The system operates with two consecutive windows, the *reference* window, followed by the *latest* window. During the initial stage of the anomaly detection process, the algorithm learns the mass profile of data in the reference window. Then, the learned profile is used to infer the anomaly scores of new data subsequently arriving in the latest window—new data that fall in high-mass subspaces is construed as normal, whereas data in low-mass or empty subspaces is interpreted as anomalous. As new data arrive at the latest window, the new mass profile is also recorded. When the latest window is full, the newly recorded profile is used to override the old profile in the reference window; thus the reference window will always store the latest profile that can be used to score the next batch of newly arriving data. Once this is done, the latest window erases its stored profile and get ready to capture profile of the next batch of newly arriving data. This process continues as long as the stream exists.

3.2 Half-Space Trees

Definition An HS-Tree of depth h is a full binary tree consisting of $2^{h+1} - 1$ nodes, in which all leaves are at the same depth, h.

When constructing a tree, the algorithm expands each node by picking a randomly selected dimension, q, in the *work space* (to be described later in this section) associated with the node. Using the mid-point of q, the algorithm bisects the work space into two half-spaces, thus creating the left child and right child of the node. Node expansion continues until the maximum depth (i.e., h or *maxDepth*) of all nodes is reached.

Each node records the mass profile of data in a work space that it represents, and has the following elements: (i) arrays min and max, which respectively store the minimum and maximum values of each dimension of the work space represented by the node; (ii) variables r and l, which record the mass profiles of data stream captured in the reference window and latest window, respectively; (iii) variable k, which records the depth of the current node; and (iv) two nodes representing the left child and right child of the current node, each associated with a half-space after the split. Figure 1 depicts an example window of (two-dimensional) data that is partitioned by a simple HS-Tree.

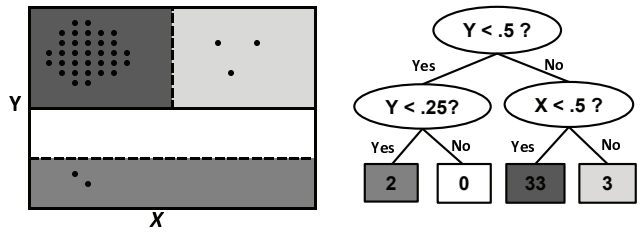


Figure 1: An example of data (in a window) partitioned by a simple HS-Tree.

Creating diverse HS-Trees is crucial to the success of our ensemble method. This is achieved by using a procedure,

Initialise Work Space, right before the construction of *each tree*. Assume that attributes' ranges are normalised to [0, 1] at the outset. Let s_q be a real number *randomly and uniformly* generated from the interval [0, 1]. A work range, $s_q \pm 2 \cdot max(s_q, 1 - s_q)$, is defined for every dimension q in the feature space D. This produces a work space that is a random perturbation of the original feature space. Since each HS-Tree is built from a different work space (defined as min and max in Algorithm 1), the result is an ensemble of diverse HS-Trees.

Algorithm 1 : BuildSingleHS-Tree(min, max, k)

Inputs: min & max - arrays of minimum and maximum values for every dimension in a Work Space,
k - current depth level
Output: an HS-Tree

1: **if** $k == maxDepth$ **then**
2: return *Node*($r \leftarrow 0, l \leftarrow 0$) {External node}
3: **else**
4: randomly select a dimension q
5: $p \leftarrow (max_q + min_q)/2$
6: {Build two nodes ($Left$ & $Right$) from a split into two equal-volume half-spaces.}
7: $temp \leftarrow max_q$; $max_q \leftarrow p$
8: $Left \leftarrow$ BuildSingleHS-Tree$(min, max, k + 1)$
9: $max_q \leftarrow temp$; $min_q \leftarrow p$
10: $Right \leftarrow$ BuildSingleHS-Tree$(min, max, k + 1)$
11: return *Node*($Left, Right, SplitAtt \leftarrow q$, $SplitValue \leftarrow p, r \leftarrow 0, l \leftarrow 0$)
12: **end if**

Algorithm 1 shows the procedure for building a single HS-Tree. Each internal node is formed by randomly selecting a dimension q (Line 4) to form two half-spaces; the split point is the mid-point of the current range of q. The mass variables of each node, r and l, are initialised to zero during the tree construction process.

Recording mass profile in HS-Trees. Once HS-Trees are constructed, mass profile of normal data must be recorded in the trees before they can be employed for anomaly detection. The process involves traversing every instance in a window through each HS-Tree. Algorithm 2 shows the process where instances in the reference window will update mass r; and mass l is updated using instances in the latest window. These two collections of mass values at each node, r and l, represent the data profiles in the two different windows. They are used in Streaming HS-Trees, which will be described in Section 3.3.

Algorithm 2 : UpdateMass$(x, Node,$ *referenceWindow*$)$

Inputs: x - an instance, $Node$ - a node in an HS-Tree
Output: none

1: (*referenceWindow*)? $Node.r$++ : $Node.l$++
2: **if** $(Node.k < maxDepth)$ **then**
3: Let $Node'$ be the next level of $Node$ that x traverses
4: UpdateMass$(x, Node',$ *referenceWindow*$)$
5: **end if**

Anomaly score. Mass in every partition of an HS-Tree is used to profile the characteristics of data. Let $m[i]$ be the mass in a half-space partition at depth level i of an HS-Tree. Under uniform mass distribution, mass values between any two partitions at levels i and j are related as follows: $m[i] \times 2^i = m[j] \times 2^j$. When the distribution is non-uniform, the following inequality establishes an ordering between partitions at different levels: $m[i] \times 2^i < m[j] \times 2^j$. We use this property to rank anomalies.

Let $Score(x, T)$ be a function that traverses a test instance x from the root of an HS-Tree (T) until a *terminal node*. This function then returns the anomaly score of x by evaluating $Node^*.r \times 2^{Node^*.k}$, where $Node^*.k$ being the depth level of the terminal node containing $Node^*.r$ instances. Here, a terminal node, or $Node^*$, is a node that has reached the maximum depth, or a node that contains $sizeLimit$ instances or fewer.

The final score for x is the sum of scores obtained from each HS-Tree in the ensemble:

$$\sum_{T \in HS\text{-}Trees} Score(x, T).$$

In practice, $sizeLimit$ is not a critical parameter, and a good default setting is 0.1ψ, where ψ is the window size. We want a large value for $maxDepth$ so that a large number of subspaces is used to capture the data profile in a comprehensive manner. But in practice, this setting is limited by the amount of computer memory available for tree construction. In our computer, we set $maxDepth$ to 15, which is adequate for capturing data stream profile. Streaming HS-Trees is able to learn data stream profile using small samples; hence, a small window size of $\psi = 250$ is sufficient for our experiments. The ensemble uses 25 trees as this is a moderate ensemble size (t) which can be easily incorporated in most machines.

3.3 Streaming HS-Trees

Algorithm 3 shows the operational procedure for Streaming HS-Trees. Line 1 builds an ensemble of Half-Space Trees. Line 2 uses the first ψ instances of the stream to record its initial reference mass profile in the HS-Trees. Since these instances come from the initial reference window, only mass r of each traversed node is updated. After these two steps, the model is ready to provide an anomaly score for each subsequent streaming point.

Mass r is used to compute the anomaly score for each streaming point (Line 8). The recording of mass for each subsequent streaming point in the latest window is then carried out on mass l (Line 9). At the end of each window, the model is updated. The model update procedure is simple—before the start of the next window, the model is updated to the latest mass by simply transferring the non-zero mass l to r (Line 14). This process is fast because it involves no structural change of the model. After this, each node with a non-zero mass l is reset to zero (Line 15).

Time and Space Complexities: The four key operations in the main loop of Algorithm 3 are: scoring (Line 8), updating mass (Line 9), model update (Line 14) and model resets (Line 15). For each of the first two operations, every

Algorithm 3 : Streaming HS-Trees(ψ, t)

Inputs: ψ - Window Size, t - number of HS-Trees
Output: s - anomaly score for each streaming instance x

Build t HS-Trees : Initialise Work Space and call Algorithm 1 for each tree
Record the first reference mass profile in HS-Trees: for each tree T, invoke UpdateMass($x, T.root$, *true*) for each item x in the first ψ instances of the stream
$Count \leftarrow 0$
while data stream continues **do**
    Receive the next streaming point x
    $s \leftarrow 0$
    for each tree T in HS-Trees **do**
        $s \leftarrow s +$ Score(x, T) {accumulate scores}
        UpdateMass($x, T.root$, *false*) {update mass l in T}
    end for
    Report s as the anomaly score for x
    $Count$++
    if $Count == \psi$ **then**
        Update model : $Node.r \leftarrow Node.l$ for every node with non-zero mass r or l
        Reset $Node.l \leftarrow 0$ for every node with non-zero mass l
        $Count \leftarrow 0$
    end if
end while

instance is traversed from a tree's root to a terminating node (i.e., $O(h)$); the last two operations each accesses at most ψ nodes but occurs $\frac{n}{\psi}$ times over the entire stream. Hence the **(average-case) amortised time complexity** for n streaming points is $O(t(h+1))$; the worst-case is $O(t(h+\psi))$, which occurs when model update and reset are performed between streaming data. These time complexities are constant when the maximum depth level (h), ensemble size (t) and the window size (ψ) are fixed.

In Streaming HS-Trees, each arriving instance is first processed and then discarded, before the next is processed. This forms a one-pass algorithm that uses a finite memory to process infinite data streams. The **space complexity** for HS-Trees is $O(t2^h)$ which is also a constant with fixed t and h.

4 Experimental Setup

Data: Columns 2 to 4 of Table 2 summarise the six large datasets used in this study. SMTP and HTTP (from KDD Cup 99) are streaming data involving network intrusions. HTTP is characterised by sudden surges of anomalies in some streaming segments. SMTP does not have surges of anomalies, but possibly exhibits some distribution changes within the streaming sequence.

In practice, it is hard to quantify whether a distribution change has indeed occurred within a stream. For this reason, we derive a dataset, SMTP+HTTP, containing the SMTP data instances follow by the HTTP data instances. We expect a distribution change to occur when the communication protocol is switched from SMTP to HTTP.

COVERTYPE is a UCI dataset [Asuncion and Newman, 2007] commonly used in data stream research. We split the anomaly class into several small groups and placed them in different segments of the dataset in order to simulate short bursts of anomalies in different streaming segments.

SHUTTLE (from UCI) and MULCROSS [Rocke and Woodruff, 1996] are datasets with little or no distribution change. However, MULCROSS contains dense clusters of anomalies that are harder to detect than scattered anomalies.

Experimental Settings: The parameter settings for Streaming HS-Trees have been discussed earlier in Section 3. In addition, all the methods are implemented in Java and all experiments were conducted on a 3GHz Pentium CPU with 1GB RAM.

Once the anomaly scores for all instances (of a segment in the data stream or of the entire dataset) were obtained, the instances were ranked based on their anomaly scores. From this ranking and the ground truth, we then computed the AUC (Area Under receiver operating characteristic Curve) [Hand and Till, 2001] to measure the performance of all anomaly detectors reported in this paper.

In all experiments, we conducted 30 independent runs of each algorithm on each dataset, and then computed the average results. A t-test at 5% level of significance was used to compare performance levels of the algorithms.

5 Experimental Results

We report the results of the experiments in this section. First, we assess the effectiveness of model adaptation to varying data distribution. This is done by comparing Streaming HS-Trees that performs regular updates of its model (denote this as HSTa), versus Streaming HS-Trees without model update (we denote this model as HSTn, which only learns from the first ψ instances of the stream).

Model Adaptation Performance: Unlike HSTa, Table 2 (Columns 5 and 6) shows that HSTn only works well in datasets with no change in distribution (i.e., SHUTTLE and MULCROSS). It performs poorly when a distribution change occurs within the data (e.g., SMTP, SMTP+HTTP, and COVERTYPE). These results are consistent with Figure 2, where HSTn degrades when there are changes in certain streaming segments of SMTP, SMTP+HTTP, and COVERTYPE. Using two artificial datasets, we also confirm that HSTn does not work well when there is a drift in normal data, whereas HSTa works well when there is a drift in either anomalies or normal data. Details of the artificial data are omitted here due to space constraints.

Comparison with Hoeffding Trees: Here, we compare HSTa with Hoeffding Trees (HT) as well as HT with Online Coordinate Boosting (BoostHT). For HT and BoostHT, we use the probability of predicting a negative class (i.e., a normal point) as the anomaly score—a true anomaly generally gets a low prediction probability, while a normal point gets a high probability. This serves as a ranking measure for anomaly detection. We employ the Java implementations of HT and BoostHT developed by Bifet *et al.* [2009].

In terms of the overall AUC scores (c.f. Columns 6 to 8 of Table 2), we expect Hoeffding Tree (HT) to produce the

Dataset	Data Size	Dimensionality	Anomaly	AUC HSTn	AUC HSTa	AUC HT	AUC BoostHT	Runtime HSTa	Runtime HT
HTTP	567497	3	attack (0.4%)	.982	.996	.994	**.998**	48	227
SMTP	95156	3	attack (0.03%)	.740	.875	.858	.692	10	39
SMTP+HTTP	662653	3	attack (0.35%)	.387	.996	.991	.993	57	272
COVERTYPE	286048	10	outlier (0.9%)	.854	.991	.998	.968	25	124
MULCROSS	262144	4	2 dense clusters (10%)	.998	.998	**1.00**	**1.00**	26	114
SHUTTLE	49097	9	class 2,3,5-7 (7%)	.999	.999	.991	.984	6	21

Table 2: Average AUC scores for Streaming HS-Trees with no model updates (HSTn) and with regular model updates (HSTa), Hoeffding Tree (HT), and HT with boosting (BoostHT). Using HSTa as a reference, scores lower than HSTa are underlined, and scores higher than HSTa are printed in boldface. Runtime is measured in seconds. The figures in brackets are the proportions of anomalies.

best overall results because it is an oracle-informed *multi-class classifier*—it is given the advantage of using the actual positive and negative class labels for training, and this is done *immediately* after each new instance is scored. In contrast, Streaming HS-Trees with regular model updates (HSTa) uses only normal data for training.

Because HT is an optimistic baseline, any one-class anomaly detector for evolving data streams that performs comparably to HT shall be deemed as a competitive method. Interestingly, Table 2 shows that HSTa actually gives higher AUC scores on four (i.e., HTTP, SMTP, SMTP+HTTP, and SHUTTLE) out of six datasets tested, as compared to HT. This observation is further examined in Figure 2, which shows that HSTa surprisingly outperforms HT in HTTP (segment 2), SMTP (segments 1 and 5), SMTP+HTTP (segment 1) and SHUTTLE (segment 1). HT is unable to cope with distribution changes in these segments, causing its detection performance to degrade.

Table 2 shows that BoostHT does not improve the performance of HT significantly. In fact, BoostHT performs poorly on SMTP, which is the most imbalanced dataset used in this study. This could be due to overfitting of the boosted model.

We also stress test the methods using datasets in which only 20% of the data are labelled. We find that HSTa outperforms HT on smaller datasets (namely SHUTTLE and SMTP) due to its ability to learn with fewer instances. Details of this experiment will be given in a future publication.

Runtime Performance: Hoeffding Tree has the ability to adapt its tree structure to streaming data—it uses Hoeffding Bound to decide the best splitting attribute when incrementally inducing a decision tree from a data stream. However, this flexibility comes with a price—the need to modify the tree causes HT's runtime to be four to six times slower than Streaming HS-Trees, as shown in the last two columns of Table 2.

Unlike Hoeffding Trees, Streaming HS-Trees does not modify or extend the tree structure during the streaming process, after it was first built. Even the tree construction procedure for Streaming HS-Trees is very efficient because the process requires no evaluation criteria for dimension or split point selections.

Effects of Parameters: Using four of the datasets for illustration, Figure 3(a) shows that the AUC scores of HSTa reach high values with 10 trees, and the scores improve steadily as the ensemble size increases. Using the SHUTTLE dataset as an example, Figures 3 (b) to (d) show that the effects of different parameter settings diminish as the ensemble size grows. This is due to the power of ensemble learning—while individual base learners (i.e., HS-Trees) may be weakened by non-optimal parameter settings for a problem at hand, the combination of these weak learners still produces reasonably good results.

6 Concluding Remarks

The proposed anomaly detection algorithm, Streaming HS-Trees, satisfies the key requirements for mining evolving data streams: (i) it is a one-pass algorithm with $O(1)$ amortised time complexity and $O(1)$ space complexity, which is capable of processing infinite data streams; (ii) it performs anomaly detection and stream adaptation in a seamless manner.

Our empirical studies show that Streaming HS-Trees with the regular model update scheme is robust in evolving data streams. In terms of detection accuracy, Streaming HS-Trees is comparable to the oracle-informed Hoeffding Tree (an optimistic baseline). In terms of runtime, Streaming HS-Trees outperforms Hoeffding Tree. Our results also show that the performance of Streaming HS-Trees is robust against different parameter settings.

References

N. Abe, B. Zadrozny, and J. Langford. Outlier detection by active learning. In *Proceedings of the 12th ACM SIGKDD*, 2006.

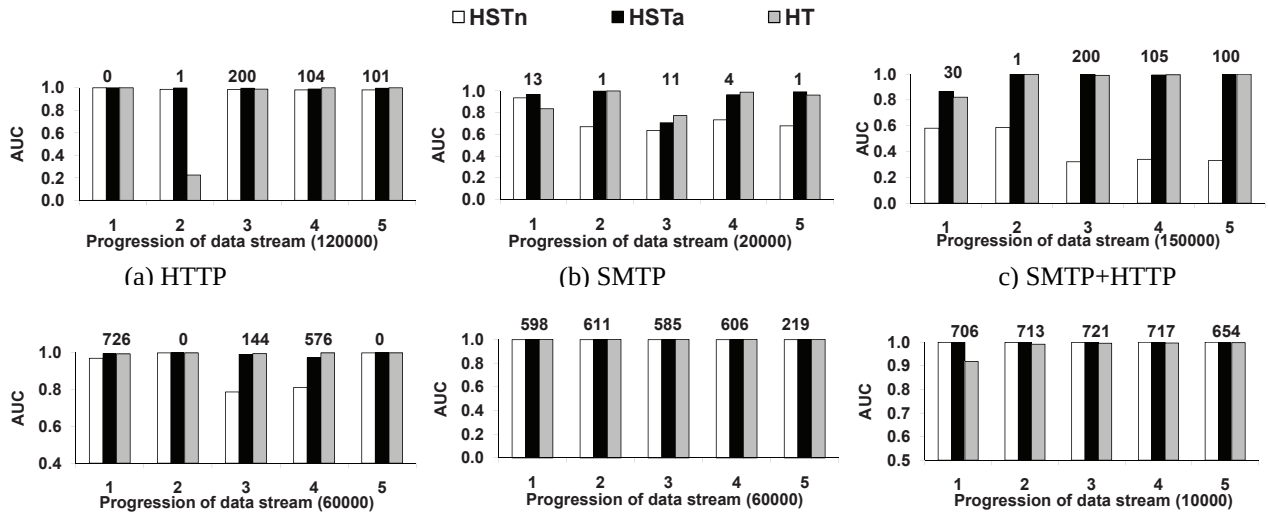
A. Asuncion and D. Newman. Uci machine learning repository. 2007.

V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley, 1994.

S.D. Bay and M. Schwabacher. Mining distance-based anomalies in near linear time with randomization and a simple pruning rule. In *Proceedings of the 9th ACM SIGKDD*, 2003.

A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD*, 2009.

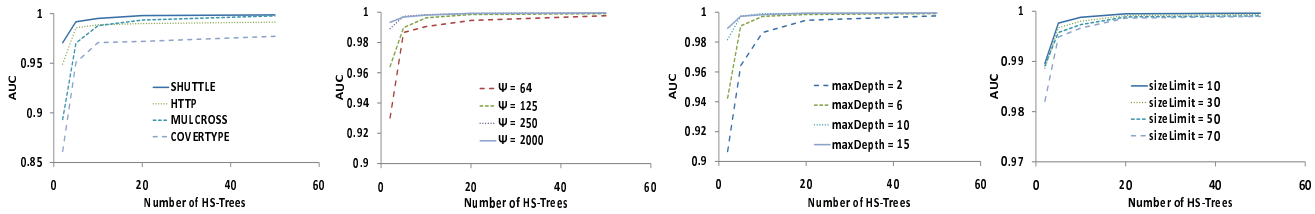
L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.



(a) HTTP (b) SMTP c) SMTP+HTTP

(d) COVERTYPE (e) MULCROSS (f) SHUTTLE

Figure 2: AUC scores in five segments. The number on top of each segment is the number of anomalies in that segment. The number in bracket is the number of items in each segment.



(a) Effect of ensemble size (t) (b) Effect of window size (ψ) (c) Effect of $maxDepth$ (d) Effect of $s0izeLimit$

Figure 3: As the ensemble size grows, the effects of different parameter settings on the detection performance tend to diminish.

M.M. Breunig, H-P. Kriegel, R.T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 6th ACM SIGKDD*, 2000.

V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41:1–58, 2009.

M. Davy, F. Desobry, A. Gretton, and C. Doncarli. An online support vector machine for abnormal events detection. *Signal Processing*, 86:2009–2025, 2005.

P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the 6th ACM SIGKDD*, 2000.

D.J. Hand and R.J. Till. A simple generalisation of the area under the roc curve for multiple class classification problems. *Machine Learning*, 45:171–186, 2001.

Z. He, X. Xu, and S. Deng. Discovering cluster-based local outliers. *Pattern Recognition Letter*, 24:1641–1650, 2003.

G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the 7th ACM SIGKDD*, 2001.

F.T. Liu, K.M. Ting, and Z.H. Zhou. Isolation forests. In *Proceedings of ICDM 2008*, 2008.

M.E. Otey, A. Ghoting, and S. Parthasarathy. Fast distributed outlier detection in mixed-attribute data sets. *Data Mining and Knowledge Discovery*, 12:203–228, 2006.

R. Pelossof, M. Jones, Vovsha, and C. I. Rudin. Online coordinate boosting. In *Proceedings of the 3rd IEEE On-line Learning for Computer Vision Workshop*, 2009.

D.M. Rocke and D.L. Woodruff. Identification of outliers in multivariate data. *Journal of the American Statistical Association*, 91:1047–1061, 1996.

B. Scholkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt. Support vector method for novelty detection. *Advances in Neural Information Processing Systems*, 12:582–588, 2002.

E.J. Spinosa, A.P. Leon, F. Carvalho, and J. Gama. Novelty detection with application to data streams. *Intelligent Data Analysis*, 13:405–422, 2009.

S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proceedings of The 32nd VLDB*, 2006.

K.M. Ting, G.T. Zhou, F.T. Liu, and S.C. Tan. Mass estimation and its applications. In *Proceedings of 16th ACM SIGKDD*, 2010.

Pattern Recognition ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Relevance feature mapping for content-based multimedia information retrieval

Guang-Tong Zhou [a], Kai Ming Ting [b], Fei Tony Liu [b], Yilong Yin [a,*]

[a] School of Computer Science and Technology, Shandong University, Jinan 250101, China
[b] Gippsland School of Information Technology, Monash University, Victoria 3842, Australia

ARTICLE INFO

Article history:
Received 22 March 2010
Received in revised form
10 July 2011
Accepted 24 September 2011

Keywords:
Content-based multimedia information retrieval
Ranking
Relevance feature
Relevance feedback
Isolation forest

ABSTRACT

This paper presents a novel ranking framework for content-based multimedia information retrieval (CBMIR). The framework introduces relevance features and a new ranking scheme. Each relevance feature measures the relevance of an instance with respect to a profile of the targeted multimedia database. We show that the task of CBMIR can be done more effectively using the relevance features than the original features. Furthermore, additional performance gain is achieved by incorporating our new ranking scheme which modifies instance rankings based on the weighted average of relevance feature values. Experiments on image and music databases validate the efficacy and efficiency of the proposed framework.

1. Introduction

We have witnessed a substantial progress in the acquisition and storage of digital media such as images, video and audio. With the rapid increase of digital multimedia collections, effective and efficient retrieval techniques have become increasingly important. Many existing multimedia information retrieval systems index and search the multimedia databases based on textual information such as keywords, surrounding text, etc. However, the text-based search suffers from the following inherent drawbacks [1,2]: (i) the textual information is usually nonexistent or incomplete with the emergence of massive multimedia databases; (ii) the textual description is not sufficient for depicting subjective semantics since different people may describe the content in different ways; and (iii) some media contents are difficult to be described in words.

To address these problems, content-based multimedia information retrieval (CBMIR) is proposed and has attracted a lot of research interest in recent years [1,3–6]. In a typical CBMIR setting, a user poses a query instance to the system in order to retrieve relevant instances from the database. However, due to the semantic gap [3,4] between high-level concepts and low-level features, the list returned by the initial search may not be good enough to satisfy the user's requirement. Thus, relevance feedback [7,8] is usually employed to allow the user to iteratively refine the query information by labeling a few positive instances as well as negative instances in each feedback round.

The performance of a CBMIR system relies on the accuracy of its ranking results. Thus, ranking is the central problem in CBMIR, and many researchers have endeavored to design a fast and effective ranking method [1,4,5]. A key ingredient in ranking is the measure used for comparing instances in the database with respect to the query. Many existing methods (e.g., [9–11,2]) use distance as the core ranking measure.

This paper presents a novel ranking framework for CBMIR that does not use distance as the ranking measure, which is fundamentally different from the above-mentioned methods. Our framework uses some form of ranking models to produce a relevance feature space. It first builds a collection of ranking models and the output of each model forms a relevance feature. Then, the models are used to map every instance from the original feature space to a new space of relevance features. Finally, the ranking and retrieval of instances, based on one query and relevance feedbacks, are computed in the new space using our proposed ranking scheme, which ranks instances based on the weighted average of relevance feature values.

Our analysis shows that the power of the proposed framework derives primarily from the relevance features and secondarily from the ranking scheme. The framework has linear time and

* Corresponding author. Tel./fax: +86 531 88391367.
E-mail addresses: zhouguangtong@gmail.com (G.-T. Zhou), kaiming.ting@monash.edu (K.M. Ting), tony.liu@monash.edu (F.T. Liu), ylyin@sdu.edu.cn (Y. Yin).

0031-3203/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2011.09.016

space complexities with respect to the database size. The on-line processing time is constant when the number of relevance features is fixed, no matter how many original features are used to represent an instance. These characteristics enable the proposed framework to scale up to large databases. In addition, our framework has a good tolerance to irrelevant features.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces our framework, followed by a detailed description in Section 4. Section 5 reports empirical studies, and Section 6 discusses related issues. Finally, this paper concludes in Section 7.

2. Related work

Many ranking methods employ distance as the core ranking measure [1,4,5]. In the case of retrieval with one query without relevance feedback, the majority of previous works have focused on different variants of distance metrics. The simplest way is to use a single distance metric, e.g., Euclidean distance or Manhattan distance. Here instances that lie near to a given query are ranked higher than instances far away from the query. However, these distance metrics are global measures and they might not produce the best results for all queries. Thus, researchers have investigated distance metrics that can be tailored to each query. For example, based on the manifold ranking algorithm [12], He et al. [9] have proposed the MRBIR method which implicitly learns a manifold metric to produce rankings.

In relevance feedback, the additional information provided by the user offers more flexibility in the design of effective ranking methods. Here the query and positive feedbacks are usually considered as positive instances, and negative feedbacks are negative instances. The refinement can be done in three ways. First, the distance metric for the initial query session can be refined based on pair-wised distance constraints derived from positive and negative instances. Commonly used techniques include distance metric learning [13,14], kernel learning [15], and manifold learning [16,17].

Second, instead of refining the distance metric, we can also tackle the problem by designing appropriate ranking schemes. For example, MARS (Multimedia Analysis and Retrieval System) [18] employs a query-point movement technique which estimates the "ideal query point" by moving it towards positive instances and away from negative ones. The ranking is produced by measuring distance with respect to the ideal query after the movement. Giacinto and Roli [10] proposed the InstRank method based on the idea that an instance is more likely to be relevant if its distance to the nearest positive instance is small, while an instance is more likely to be irrelevant if its distance to the nearest negative instance is small. Qsim [11] advocates ranking instances based on the query-sensitive similarity measure, which takes into account the queried concept when measuring similarities. Note that these methods are all based on some predefined or learned distance metrics.

Third, some methods transform the CBMIR problem into a classification problem, and solve it using classification techniques such as support vector machine [19] and Bayesian method [2]. A representative method called BALAS [2] first estimates the probability density function of positive and negative classes, and then the ranking is produced within a Bayesian learning framework. However, most classification methods are designed to classify instances into a fixed number of classes and are not designed for ranking instances. Thus, the ranking results might be suboptimal.

This paper proposes to rank instances through a new framework that does not require distance calculation—a computationally expensive process. This is fundamentally different from most existing methods. Our framework is able to deal with retrieval tasks with one query as well as in relevance feedback. In contrast, most of the above-mentioned methods were designed to be used in relevance feedback only, e.g., InstRank, Qsim and BALAS.

Note that meta-search [20] employs an ensemble of ranking models for information retrieval. However, this technique aims at improving the retrieval performance by combining the ranking results returned by multiple search engines. This is a different problem from the one we addressed. It is also worth noting that Rasiwasia et al. [21] proposed the query-by-semantic-example method which maps and retrieves instances in a semantic space. Here a set of semantic-level concepts has to be predefined in order to construct the semantic features. On the contrary, the relevance features used in this paper are automatically generated—users do not need to specify them.

3. The proposed framework

Generally speaking, a CBMIR system is composed of four parts [22]: (i) a given multimedia database $\mathcal{D}$; (ii) a query $\mathcal{Q}$; (iii) a model $\mathbb{F}(\mathcal{Q},\mathcal{D})$ to model the relationships between instances in $\mathcal{Q}$ and $\mathcal{D}$; and (iv) a ranking scheme $R(\mathcal{D}|\mathcal{Q})$ which defines an ordering among the database instances with respect to $\mathcal{Q}$. On the other hand, a ranking system consists of three components: (i) a given data set $\hat{\mathcal{D}}$; (ii) a model $\hat{\mathbb{F}}(\hat{\mathcal{D}})$ to model the relationships between instances in $\hat{\mathcal{D}}$; and (iii) a ranking scheme $\hat{R}(\hat{\mathcal{D}})$ which produces an ordering for all the instances in $\hat{\mathcal{D}}$. Ranking in CBMIR are typically provided by distance metrics. In this work, we show an alternative method, that is more suitable for CBMIR, using an ensemble of ranking systems.

Here, we propose to map the database $\mathcal{D}$ from the original d-dimensional feature space $\mathbb{R}^d$ into a new space $\mathbb{R}^t$ to form a new database $\mathcal{D}'$ by using an ensemble of t ranking models, i.e., $\tilde{\mathbb{F}}=[\hat{\mathbb{F}}_1,\hat{\mathbb{F}}_2,\ldots,\hat{\mathbb{F}}_t]$. Each ranking model is regarded as a feature descriptor, and the ranking output is the feature value; for an instance, the t ranking outputs from the t ranking models constitute the new t-dimensional feature vector. Given a query $\mathcal{Q}$, we first map it into the new space to obtain $\mathcal{Q}'$, and then we employ a ranking scheme $R'(\mathcal{D}'|\mathcal{Q}')$ to rank the instances in $\mathcal{D}'$. Note that R' can be any existing ranking scheme. But we propose a new ranking scheme based on the weighted average of relevance feature values to avoid the costly distance or similarity calculation. We show in this paper that the ensemble of ranking models, i.e., $\tilde{\mathbb{F}}$, can be implemented using an anomaly detector called Isolation Forest, or iForest [23].

iForest builds an ensemble of isolation trees (or iTrees) to detect anomalies. Each iTree is constructed on a fix-sized random sub-sample of the given data set. The tree growing process recursively random-partitions the sub-sample along axis-parallel coordinates until every instance is isolated from the rest of the instances or a specified height limit is reached. Each iTree is a ranking model which describes a data profile from the view of the underlying sub-sample and produces a ranking output in terms of path length for any test instance. The ranking output can be interpreted as: a short path length indicates irrelevance to the profile because an instance, which has different data characteristics from the majorities, is easily isolated by a few random partitions; on the other hand, a long path shows relevance to the profile. For anomaly detection tasks, instances identified to be irrelevant to the various profiles modeled by a number of iTrees are deemed to be anomalies, and instances relevant to the profiles are normal points. The algorithms to produce iTree and iForest are provided in Appendix A.

In our framework, we first build an iForest, which is composed of t iTrees, to map instances from the original feature space to the

Please cite this article as: G.-T. Zhou, et al., Relevance feature mapping for content-based multimedia information retrieval, Pattern Recognition (2011), doi:10.1016/j.patcog.2011.09.016

relevance feature space, i.e., $\mathbb{R}^d \rightarrow \mathbb{R}^t$. Different iTrees profile different aspects of the multimedia database. We treat each iTree as a feature descriptor, and the feature value (i.e., path length) is a measure of relevance with respect to the profile modeled by the iTree. The representation of an instance in the new space is a vector of relevance features; hence the name *relevance feature mapping*. To implement $R'(\mathcal{D}'|\mathcal{Q}')$, we have also designed a new ranking scheme based on the weighted average of relevance feature values. We call our framework `ReFeat` which refers to the retrieval based on *Re*levance *Feat*ure mapping.

4. ReFeat

`ReFeat` has two stages. The first off-line modeling stage builds an iForest to perform relevance feature mapping and the second on-line retrieval stage ranks instances with respect to the query. We first describe the two stages in the next two subsections, followed by explaining why our ranking scheme works in Section 4.3. We then provide our treatment for relevance feedback in Section 4.4. The algorithmic complexity is analyzed in the last subsection.

4.1. Off-line modeling and relevance feature mapping

In off-line modeling, we build an iForest from the given database $\mathcal{D}$. Here t iTrees are constructed, each built on a sub-sample of randomly selected ψ instances from $\mathcal{D}$. After iForest is built, $\mathcal{D}$ is mapped to $\mathcal{D}'$ as follows.

Let $\ell_i(\mathbf{x})$ denotes the path length of an instance $\mathbf{x} \in \mathcal{D}$ on an iTree T_i ($i \in \{1, 2, \ldots, t\}$). We map $\mathbf{x}$ to the relevance feature space as: $\mathbf{x}' = [\ell_1(\mathbf{x}), \ell_2(\mathbf{x}), \ldots, \ell_t(\mathbf{x})]^T$. All the instances in $\mathcal{D}$ are mapped through the relevance feature mapping to form a new database $\mathcal{D}' = \{\mathbf{x}' | \forall \mathbf{x} \in \mathcal{D}\}$. Note that this stage does not require any user intervention. Thus, $\mathcal{D}'$ is generated off-line to accelerate the following on-line retrieval process.

4.2. On-line retrieval with one query

Given a query instance $\mathbf{q}$, `ReFeat` maps it to $\mathbf{q}' = [\ell_1(\mathbf{q}), \ldots, \ell_t(\mathbf{q})]^T$. To retrieve instances relevant to $\mathbf{q}$, we first assign a weight to each feature due to $\mathbf{q}$: a high weight is assigned to a feature which signifies that $\mathbf{q}$ is relevant to the profile modeled by the feature; otherwise, a low weight is assigned. Then the ranking score for every instance in the database is computed using a weighted average of its relevance feature values. The instances having the highest scores are regarded to be the most relevant to the query. To implement this, we define a weight for feature i as:

$$w_i(\mathbf{q}) = \frac{\ell_i(\mathbf{q})}{c(\psi)} - 1. \tag{1}$$

$c(\psi)$ is a normalization term which estimates the average path length of a ψ-sized iTree. The $c(\cdot)$ function is defined as follows [23]:

$$c(n) = \begin{cases} 2(\ln(n-1) - (n-1)/n + E) & \text{if } n > 1, \\ 0 & \text{if } n = 1, \end{cases} \tag{2}$$

where $E \approx 0.5772$ is the Euler's constant.

Finally, the ranking score of an instance $\mathbf{x}$ with respect to the query $\mathbf{q}$ is given by the weighted average of feature values:

$$Score(\mathbf{x}|\mathbf{q}) = \frac{1}{t}\sum_{i=1}^{t}(w_i(\mathbf{q}) \times \ell_i(\mathbf{x})). \tag{3}$$

Eq. (3) gives high scores to instances which have long path lengths on many highly weighted features induced by the query; and it produces low scores to instances which have short path lengths on many lowly weighted features. $Score(\mathbf{x}|\mathbf{q})$ can be negative. If required, strictly positive scores can be produced by using an exponential mapping. For the rest of this paper, we refer to the ranking based on the weighted average of feature values as our ranking scheme.

It is worth noting that the off-line modeling of iForest utilizes no distance or similarity measure [23], and the proposed on-line ranking scheme also avoids distance or similarity calculation through Eqs. (1) and (3). This characteristic differentiates `ReFeat` from most existing methods which are based on certain distance or similarity measures.

4.3. Understanding the ranking scheme

Our ranking scheme is based on the idea that similar instances share many relevance features with long path lengths from iTrees; whereas dissimilar instances have many relevance features with short path lengths.

A region defined by a long path length in an iTree has many same splitting conditions, where each condition is defined by an internal node along the path from the root to the external node. Thus, intuitively, instances falling into each of these regions (defined by long path lengths) are likely to be more similar than those instances falling into other regions. This explains why we use Eq. (1) to assign high weights to iTrees where the query is estimated to have long path lengths—a big contribution to the relevance score through Eq. (3) if the test instance also achieves long path lengths on these iTrees. On the other hand, if an instance is estimated to have a short path length on an iTree, then it is most likely to be different from the instances falling into the regions defined by long-path-length external nodes. Thus, Eq. (1) assigns negative weights to the iTrees in which the query has short path lengths—via Eq. (3) to penalize the test instances with long path lengths in these iTrees. In addition, if the query is estimated to have a path length around $c(\psi)$, then we simply assign a small or zero weight because instances having similar path lengths are likely to be in different regions.

In the following paragraphs, we first provide the topologically distinct iTree structures in the setting we have used in our experiment. Then, we show that the majority of iTrees produced from a database have distinct long and short path lengths that allow our scheme to identify similar instances from dissimilar ones through ranking.

The parameters we have used in the experiment are: the sub-sample size $\psi = 8$ and the height limit $h = \lceil \log_2 \psi \rceil = 3$. This produces a total of 17 topologically distinct tree structures as shown in Fig. 1. To obtain the path length of an instance $\mathbf{x}$ from an iTree, $\mathbf{x}$ traverses from the root of the iTree to an external node; and the path length is computed as the number of edges traversed plus the estimated average path length of an unbuilt subtree from a sample of *Size* instances which is $c(Size)$, where *Size* is the number of sub-sample instances at the external node and $c(\cdot)$ is defined in Eq. (2). Note that out of the 17 structures depicted in Fig. 1, structures (a)–(g) all have the minimum path length equal to 1; and structures (h)–(p) have the minimum path length equal to 2. These structures have the maximum path lengths vary from $3+c(5)$, $3+c(4)$, $3+c(3)$ to $3+c(2)$. Only structure (q) is a balanced tree which gives the same path length for all instances.

An iTree is only useful if it is imbalanced and provides long and short path lengths that differentiate similar and dissimilar instances. It is also preferred to have the maximum path length in only one external node that uniquely identifies the neighborhood region. A total of 10 structures, i.e., (a)–(f) and (h)–(k), satisfy this essential property,[1] where the maximum path lengths are

[1] Structure (d) is an exception but it still stipulates the neighborhood region by at least two splitting conditions. We include (d) here to facilitate the following analysis.

Please cite this article as: G.-T. Zhou, et al., Relevance feature mapping for content-based multimedia information retrieval, Pattern Recognition (2011), doi:10.1016/j.patcog.2011.09.016

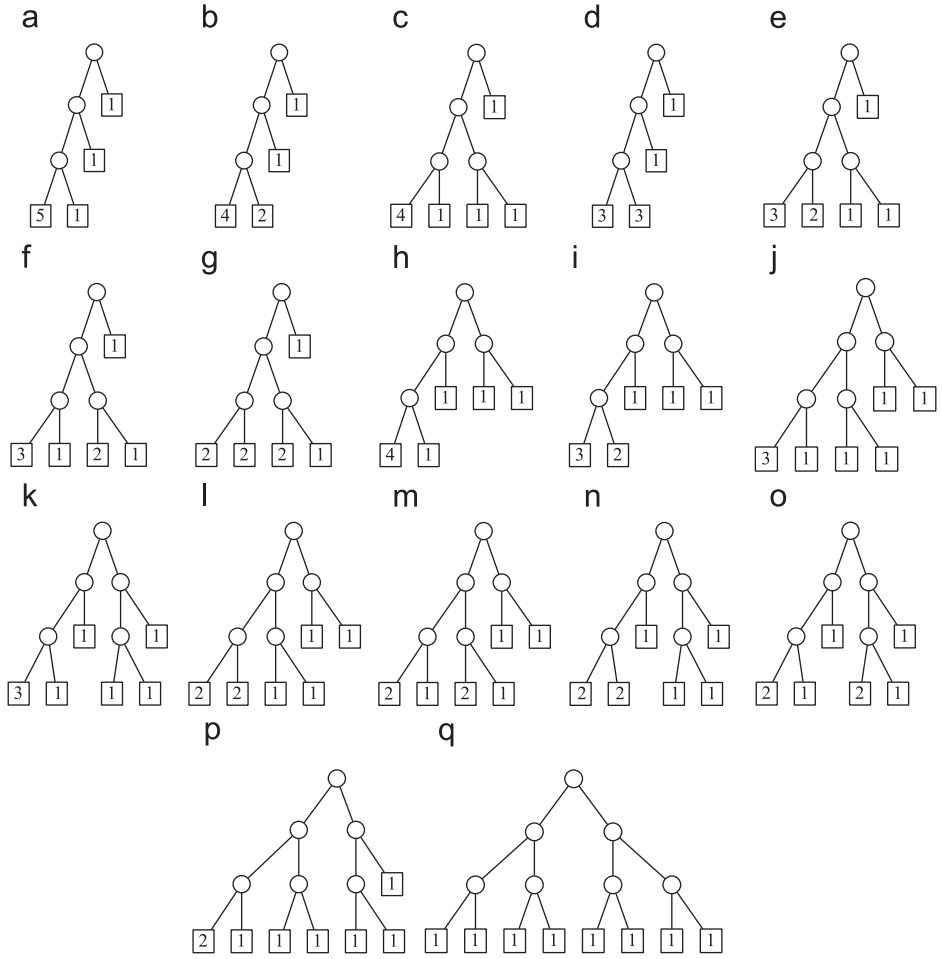


Fig. 1. The 17 unique iTree structures with $\psi = 8$ and $h = 3$. Circles (○) denote internal nodes, and squares (□) are external nodes. The figure in an external node indicates the number of sub-sample instances split in the node, i.e., the "*Size*" of the node.

$3+c(5)$, $3+c(4)$ and $3+c(3)$. A total of six structures, i.e., (g) and (l)–(p), are also good by providing short path lengths. An iTree like structure (q) which gives the same path length for all instances is useless for our purpose.

We employ δ, which is the difference between the maximum path length and the minimum path length of an iTree, to indicate how imbalance the iTree is. Out of the 17 topologically distinct tree structures, we have only eight δ values: 0, $1+c(2)$, $2+c(2)$, $1+c(3)$, $1+c(4)$, $2+c(3)$, $2+c(4)$, and $2+c(5)$, which range from balanced tree (q) to highly imbalanced tree (a).

Using the COREL image database [24], we generate 1000 iTrees and then tally the number of trees for each δ value. Fig. 2(a) shows the result: more than 75% of the iTrees have $\delta \geq 1+c(3)$ which represents the 10 imbalanced iTree structures (a)–(f) and (h)–(k). The near-balanced trees (having $0 < \delta \leq 2+c(2)$) constitute about 23% of the iTrees which represents the six structures (g) and (l)–(p). The balanced iTrees constitute less than 1%. The result shows that the majority of the generated iTrees are useful for identifying similar instances as well as dissimilar instances.

To further enhance the understanding, we provide statistics of the path lengths in the following case study. We select a rose image (Fig. 2(b)) from the COREL database as a query. Another rose image (Fig. 2(c)) is considered as relevant, and a beach image (Fig. 2(d)) is treated as irrelevant. We estimate the path lengths of the three images on the above-generated 1000 iTrees. Considering the 17 distinct iTree structures, we have seven possible path length values ranging from the longest to the shortest: $3+c(5)$, $3+c(4)$, $3+c(3)$, $3+c(2)$, 3, 2, and 1. We then divide the 1000 iTrees into seven categories based on the query's path lengths. In each category, we calculate the proportion of iTrees that produce different path lengths for the relevant image and the irrelevant image, and the results are provided in Table 1. It shows that: on highly weighted iTrees (in which the query has long path lengths, shown in top rows in Table 1(a) and (b)), the relevant image has significantly more long path lengths than the irrelevant image; on negatively weighted iTrees (in which the query has short path lengths, shown in bottom rows in Table 1(a) and (b)), the relevant image has noticeably less long path lengths. This explains why the relevant image scores larger than the irrelevant one through Eq. (3) in our ranking scheme. In this case, the scores for relevant and irrelevant images are 1.14 and 0.89, respectively.

Also notice that the similarity between the relevant image and the query is implied by the high proportion of iTrees when the path length is matched between the two images (see the numbers in the diagonal of Table 1(a)). The corresponding proportions of iTrees are significantly less between the irrelevant image and the query image, shown in Table 1(b).

4.4. On-line retrieval in relevance feedback

If feedbacks are available, we use them to refine the retrieval result by modifying the feature weights. Here the query is denoted by $\mathcal{Q} = \mathcal{P} \cup \mathcal{N}$, where $\mathcal{P}$ is the set of positive feedbacks and the initial query; and $\mathcal{N}$ is the set of negative feedbacks. Begin with the initial query $\mathbf{q}$, they are initialized as follows: $\mathcal{P} = \{\mathbf{q}\}$ and

Please cite this article as: G.-T. Zhou, et al., Relevance feature mapping for content-based multimedia information retrieval, Pattern Recognition (2011), doi:10.1016/j.patcog.2011.09.016

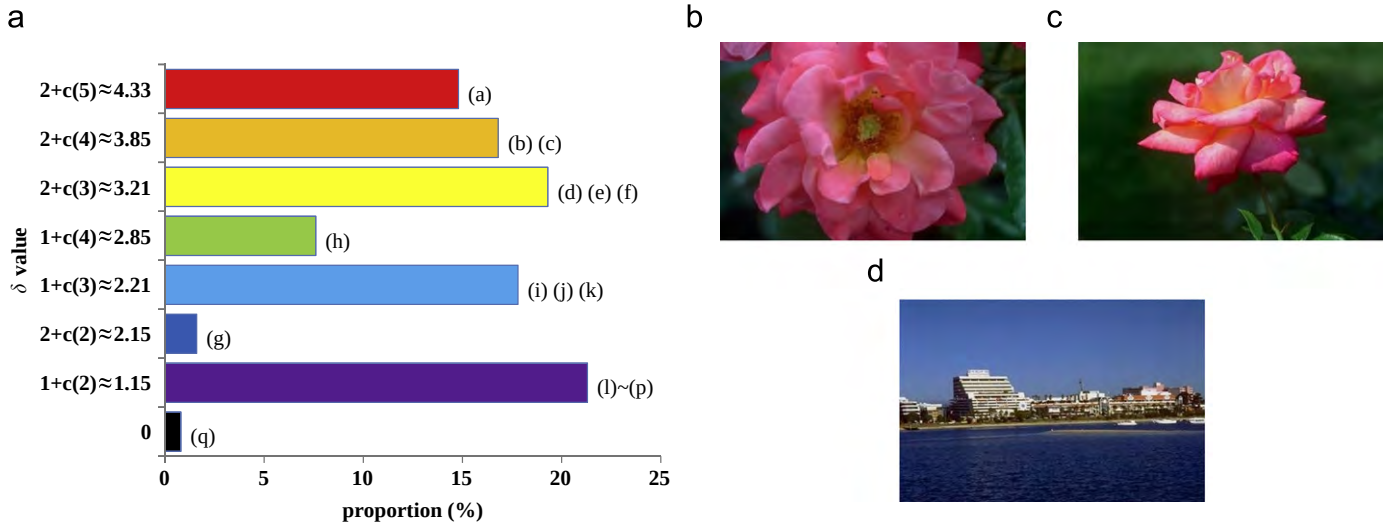


Fig. 2. Statistics of iTrees and the sample images used in our case study. (a) The proportions of 1000 iTrees with different δ values. (b) Query image. (c) Relevant image. (d) Irrelevant image.

Table 1
The proportion (%) of iTrees that produce different path lengths for the relevant image (Fig. 2(c)) and the irrelevant image (Fig. 2(d)) out of the number of iTrees that estimate a specified path length for the query (Fig. 2(b)). For this query image, the numbers of iTrees having path lengths $3+c(5)$, $3+c(4)$, $3+c(3)$, $3+c(2)$, 3, 2, 1 are 79, 99, 121, 162, 268, 189, 82, respectively.

Query's path length	Proportion of iTrees with path length						
	$3+c(5)$	$3+c(4)$	$3+c(3)$	$3+c(2)$	3	2	1
(a) Relevant image							
$3+c(5)$	**83.5**	N/A	N/A	N/A	3.8	7.6	5.1
$3+c(4)$	N/A	**77.8**	N/A	3.0	8.1	4.0	7.1
$3+c(3)$	N/A	N/A	**73.6**	8.3	7.4	7.4	3.3
$3+c(2)$	N/A	1.2	5.6	**74.1**	9.3	5.6	4.3
3	1.9	1.5	7.1	6.3	**75.7**	4.5	3.0
2	2.1	3.2	4.2	4.8	7.4	**77.2**	1.1
1	4.9	4.9	3.7	2.4	3.7	2.4	**78.0**
(b) Irrelevant image							
$3+c(5)$	29.1	N/A	N/A	N/A	19.0	15.2	**36.7**
$3+c(4)$	N/A	**33.3**	N/A	7.1	21.2	23.2	15.2
$3+c(3)$	N/A	N/A	**29.8**	8.3	20.7	26.4	14.9
$3+c(2)$	N/A	3.1	6.2	24.7	**32.7**	21.0	12.3
3	1.9	7.8	12.7	17.9	**29.9**	19.0	10.8
2	9.0	11.1	14.8	15.3	**23.8**	18.5	7.4
1	17.1	15.9	11.0	7.3	**29.3**	9.8	9.8

$\mathcal{N}=\emptyset$. Then, $\mathcal{P}$ and $\mathcal{N}$ are enriched with the instances labeled by the user in the relevance feedback process.

If only positive feedbacks are provided, `ReFeat` puts them in $\mathcal{P}$ and calculates the feature weights in the same way as that for the initial query. Formally, `ReFeat` defines the weight of feature i due to a positive feedback $\mathbf{z}^+ \in \mathcal{P}$ as:

$$w_i^+(\mathbf{z}^+) = \frac{\ell_i(\mathbf{z}^+)}{c(\psi)} - 1. \tag{4}$$

Then the resultant weight for feature i due to $\mathcal{P}$ is obtained by averaging the weights produced by all the positive instances in $\mathcal{P}$:

$$w_i^+(\mathcal{P}) = \frac{1}{|\mathcal{P}|}\sum_{k=1}^{|\mathcal{P}|} w_i^+(\mathbf{z}_k^+). \tag{5}$$

Here $|\cdot|$ denotes the size of a set. Now by replacing $w_i(\mathbf{q})$ with $w_i^+(\mathcal{P})$ in Eq. (3), a new ranking score can be produced for each instance and a refined retrieval result is returned to the user.

When negative feedbacks are also provided in relevance feedback, `ReFeat` puts them in $\mathcal{N}$ and defines the weight in an opposite way as for the initial query: a high weight is assigned to a feature which signifies that a negative feedback is irrelevant to the profile modeled by the feature; otherwise, a low weight is assigned. To implement this, `ReFeat` calculates the weight for feature i due to a negative feedback $\mathbf{z}^- \in \mathcal{N}$ as:

$$w_i^-(\mathbf{z}^-) = 1 - \frac{\ell_i(\mathbf{z}^-)}{c(\psi)}. \tag{6}$$

The resultant weight for feature i due to $\mathcal{N}$ is generated by averaging over all negative instances in $\mathcal{N}$:

$$w_i^-(\mathcal{N}) = \frac{1}{|\mathcal{N}|}\sum_{s=1}^{|\mathcal{N}|} w_i^-(\mathbf{z}_s^-). \tag{7}$$

Now the final weight for feature i can be obtained by aggregating $w_i^+(\mathcal{P})$ and $w_i^-(\mathcal{N})$. The aggregation can be realized in different ways. Here we use a simple summing method: $w_i(\mathcal{Q}) = w_i^+(\mathcal{P}) + \gamma w_i^-(\mathcal{N})$, where $\gamma \in (0,1]$ is a trade-off parameter accounting for the relative weights of the contributions between positive and negative instances. It is reasonable that positive instances make more contribution to the final ranking than negative ones. Since the farther an instance lies from positive instances, the less likely that it is a relevant one. However, we can not draw an opposite conclusion for negative instances: if an instance lies far from negative instances, it is not necessarily relevant, since it may be far from positive instances too. Similar strategies were employed in previous works (e.g., [9,11]). The empirical study presented in Section 5.2.6 also shows the efficacy of this strategy.

Finally, `ReFeat` estimates the ranking score for every instance in the database using Eq. (3) (by replacing $w_i(\mathbf{q})$ with $w_i(\mathcal{Q})$), and returns the instances by ranking them in a descending order according to their scores.

4.5. Complexity

We now analyze the time complexity of `ReFeat`. In the off-line modeling stage, building the iForest model takes $O(t\psi \log \psi)$ and the mapping from $\mathcal{D}$ to $\mathcal{D}'$ costs $O(|\mathcal{D}|t \log \psi)$ [23]. Thus, the total time complexity is $O((|\mathcal{D}|+\psi)t \log \psi)$. In the on-line retrieval stage, the relevance feature mapping for the query costs

Table 2
Time complexities of ReFeat, Euclidean, InstRank [10] and Qsim [11] for on-line retrieval. Here d is the original dimension of the multimedia database $\mathcal{D}$. InstRank and Qsim are methods dealing with relevance feedback only.

Method	With one query	In relevance feedback
ReFeat	$O((\lvert\mathcal{D}\rvert+\log\psi)\times t)$	$O((\lvert\mathcal{D}\rvert+\lvert\mathcal{Q}\rvert)\times t)$
Euclidean	$O(\lvert\mathcal{D}\rvert\times d)$	N/A
InstRank	N/A	$O(\lvert\mathcal{D}\rvert\times\lvert\mathcal{Q}\rvert\times d)$
Qsim	N/A	$O(\lvert\mathcal{D}\rvert\times\lvert\mathcal{Q}\rvert\times(d+\lvert\mathcal{P}\rvert))$

$O(t\log\psi)$, calculating weights takes $O(\lvert\mathcal{Q}\rvert t)$, and producing ranking scores for all instances in the database costs $O(\lvert\mathcal{D}\rvert t)$. Thus, for a query session, ReFeat has a time complexity of $O((\lvert\mathcal{D}\rvert+\lvert\mathcal{Q}\rvert+\log\psi)t)$. It is worth noting that $\lvert\mathcal{Q}\rvert$ is much smaller than $\lvert\mathcal{D}\rvert$, and both t and ψ are fixed at the beginning of the off-line modeling stage which do not change in on-line retrieval. Thus, ReFeat has a linear time complexity with respect to $\lvert\mathcal{D}\rvert$ in both the off-line modeling stage and the on-line retrieval stage, which makes it possible to scale up to large multimedia databases. Table 2 lists the time complexities of ReFeat as well as three other methods for on-line retrieval. It shows that ReFeat has a relatively low time complexity in on-line retrieval although it needs an additional modeling stage. Note that we also compare BALAS and MRBIR in our experiments. Although it is difficult to analyze their complexities, the experimental results show that BALAS and MRBIR usually spend much longer time than ReFeat.

The space requirement of our off-line model is also linear with respect to $\lvert\mathcal{D}\rvert$, since the database $\mathcal{D}'$ costs $O(\lvert\mathcal{D}\rvert t)$ and iForest requires $O((2\psi-1)tb)$ memory space only [23], where b is the memory size taken by a tree node.

5. Experiments

The performance of ReFeat is evaluated with content-based image and music retrieval tasks on COREL image database (which is used in [24]) and GTZAN music database [25], respectively. The image database consists of 10 000 COREL images that are collected from 100 categories such as car, forest, sunset, tiger, etc.; each category contains 100 images. As in [24], each image is represented by a 67-dimensional feature vector which consists of 32 color features generated by HSV histogram, 24 texture features derived from Gabor wavelet transformation and 11 shape features including invariant moments, center coordinates, area and principal axis orientation. The music database contains 1000 songs which are uniformly distributed over 10 genres including classical, country, disco, hiphop, jazz, rock, blues, reggae, pop, and metal. Each song is a 30-second excerpt which is stored as a 22 050 Hz, 16-bit, mono-audio file. Following the feature extraction steps in [26], we split each song into 3-second segments, where a MFCC [25] feature vector is extracted from each segment and the top 20 MFCC coefficients are kept to represent the segment. The mean and the lower-triangular covariance matrix of MFCC features are calculated and concatenated into a 230-dimensional feature vector to represent the song. Note that there is no feature selection although it may be beneficial. The same features are used by all the compared methods because we are only interested in the relative instead of absolute performance of the methods.

Our experiments study the retrieval performance of ReFeat both with one query and in relevance feedback. The initial queries are chosen as follows: for the image database, we randomly select five images from each category to obtain 500 initial queries; and for the music database, we use every song in the whole database and there are a total of 1000 initial queries. For a query, the images/songs within the same category/genre are regarded as relevant and the rest are irrelevant. We continue to perform five rounds of relevance feedback for each query. In each round, we randomly select two relevant and two irrelevant instances as positive and negative feedbacks, respectively. Note that an instance will not be considered for selection if it has been chosen as a feedback in previous rounds. To simulate different users' behavior, this relevance feedback process is repeated five times, each with a different random series of feedbacks. Finally, we report the average result over multiple runs for the initial query and the subsequent rounds of feedback.

PR-curve is a commonly used performance measure in information retrieval. It depicts the relationship between precision and recall of a retrieval system. In the experiments, we employ PR-curve to evaluate the retrieval performance with one query. However, in relevance feedback, a single PR-curve is not enough to reveal the performance changes with the increasing number of feedbacks. Thus, we use Mean Average Precision (MAP) and Precision at N (P@N) [4]. MAP is the average of precisions computed at the point of each of the relevant instances in the ranked sequence. P@N records the fraction of relevant instances in the top-N ranked instances, and we empirically set $N=50$ in the following experiments. The higher the MAP and P@N values, the better the performance. Notice that previous works (e.g., [10,11]) have included feedback instances in the evaluation of retrieval performance. However, this calculation inflates the performance since the feedbacks are labeled instances that should not be displayed to the user. Thus, we have excluded feedbacks in our performance evaluation.

The efficacy and efficiency of ReFeat are validated in the next subsection, followed by empirical studies showing the effectiveness of the relevance feature mapping, the utility of our ranking scheme, the influence of increasing database dimension, and the effect of different parameter settings in ReFeat. All the experiments are conducted on a Pentium 4 machine with a 1.86 GHz CPU and 2 GB memory.

5.1. Comparison with existing methods

In this subsection, we first compare ReFeat with the Euclidean distance based method and a manifold ranking method MRBIR [9] when no relevance feedback is performed. Then with relevance feedbacks, Qsim [11], InstRank [10], MRBIR [9] and BALAS [2] are employed for benchmarking. Here Qsim and InstRank are methods for improving ranking calculation, and BALAS is a Bayesian learning method. Because Qsim and InstRank are proposed to be used only in relevance feedback for improving similarity calculation, we employ Euclidean distance to measure the relevance so that they can deal with query without feedbacks. BALAS also does not mean to work with single query and there is no comparison of BALAS for retrieval with one query. Note that we also include the chance performance of random method (called Random) as a baseline method.

There are three parameters in ReFeat: number of relevance features t, sub-sample size ψ and trade-off parameter γ. ReFeat is not very sensitive to γ when $\gamma\in[0.1,0.4]$, and we set $\gamma=0.25$ for both the image and music databases. The values of t and ψ are problem-dependent. We set $t=1000$, $\psi=8$ for the image database, and $t=1000$, $\psi=4$ for the music database. The effect of the three parameters on the performance of ReFeat is studied in Section 5.2. For MRBIR, we keep the default parameter settings as in [9]: 200 nearest neighbors are used for constructing the weighted graphs; the contribution of negative ranking scores is weighted by 0.25; the trade-off parameter α is set to be 0.99 in the manifold ranking algorithm, which iterates 50 rounds to

Please cite this article as: G.-T. Zhou, et al., Relevance feature mapping for content-based multimedia information retrieval, Pattern Recognition (2011), doi:10.1016/j.patcog.2011.09.016

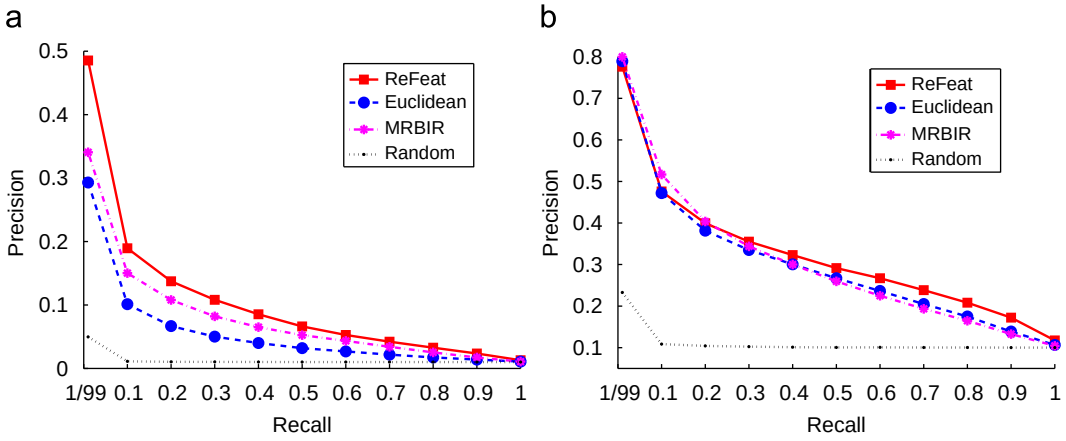


Fig. 3. PR-curves of `ReFeat`, `Euclidean`, `MRBIR` and `Random` for retrieval with one query. (a) COREL image database. (b) GTZAN music database.

obtain the final results. The only difference lies in the setting of δ_l in computing Laplacian kernels: while [9] empirically sets $\delta_l = 0.05$, we select the best δ_l from {0.0125,0.025,0.05,0.1,0.5,1} and use 0.05 for the image database and 0.025 for the music database. For `BALAS`, we generate five random instances to represent each negative feedback (in addition to the feedback instances selected from the database) to enable the estimation of the probability density function. The threshold for determining high trustworthy dimensions is kept to be 0.7 as in [2]. `Qsim` and `InstRank` do not have parameters that need to be set.

5.1.1. Retrieval with one query

The PR-curves of `ReFeat`, `Euclidean`, `MRBIR` and `Random` for retrieval with one query are presented in Fig. 3. It shows that on the image database, `ReFeat` outperforms the other three compared methods, and `MRBIR` is better than `Euclidean`; and on the music database, `ReFeat` is better than `Euclidean`, `MRBIR` and `Random` on most recall values, except that `MRBIR` achieves the best precision when the recall value ≤ 0.2.

We also provide a detailed comparison in Table 3 to gain further insight into the advantages of `ReFeat`. For each initial query, we calculate the MAP and P@50 values using every compared method, and present the average results in Table 3. A paired t-test at 5% significance level is performed for the MAP (and P@50) series over all queries, and we record the probability of rejecting the hypothesis that `ReFeat` is significantly better than the compared method. The average results in Table 3 reveal that `ReFeat` performs better than `Euclidean` and `MRBIR`, and the t-test results show that the difference is statistically significant. The only exception is that `ReFeat` achieves no significant result against `MRBIR` on the music database. These observations reveal the superior performance of `ReFeat` for retrieval with one query.

5.1.2. Retrieval in relevance feedback

Fig. 4 shows the MAP and P@50 results for retrieval in relevance feedback. Note that round 0 presents the retrieval performance with one query only, and `Euclidean` is used as the base method for `Qsim` and `InstRank`.

It is found in Fig. 4 that as the number of feedback rounds increases, the retrieval performance of most methods tends to improve. However, `BALAS` performs poor on the music database, and we suspect that this might be caused by the violation of feature independent assumption on the music database. Nevertheless, Fig. 4 clearly reflects that `ReFeat` achieves the best MAP and P@50 no matter how many feedbacks are provided. Since `ReFeat` has superior performance with both one query and relevance feedbacks, we can conclude that `ReFeat` is highly effective for CBMIR.

Table 3
A detailed comparison (average MAP ($\times 10^{-2}$), average P@50 ($\times 10^{-2}$) and t-test) of `ReFeat` against `Euclidean` and `MRBIR` for retrieval with one query.

Method	COREL image database		GTZAN music database	
	MAP	P@50	MAP	P@50
`ReFeat`	**9.11**	**15.64**	**31.06**	37.59
`Euclidean`	4.76	8.97	28.94	36.18
`MRBIR`	7.03	11.99	29.27	**37.74**
t-test results of `ReFeat` against:				
`Euclidean`	4.6×10^{-28}	1.4×10^{-29}	2.7×10^{-14}	2.0×10^{-4}
`MRBIR`	2.3×10^{-7}	2.5×10^{-9}	1.9×10^{-10}	0.7199

5.1.3. Processing time

The average on-line processing time of all compared methods is tabulated in Table 4 where the shortest time at each round is boldfaced. Note that the processing time for retrieval with one query is reported in round 0, where the time costs of `Qsim` and `InstRank` are filled by that of `Euclidean`.

Table 4 shows that `ReFeat` has the best efficiency except that it spends a bit more time than `Euclidean` for retrieval with one query on the image database. This implies that `Euclidean` prefers low-dimensional databases and `ReFeat` is more efficient on high-dimensional databases. We have provided a detailed analysis in Section 5.2.3 on how the database dimension influences the retrieving time of the compared methods. Note that `ReFeat` achieves the shortest and near constant processing time regardless of the feedback round. The time is independent of the number of feedbacks because the time complexity of `ReFeat` for retrieval in relevance feedback, i.e., $O((|\mathcal{D}|+|\mathcal{Q}|)\times t)$ (as shown in Table 2), is dominated by $O(|\mathcal{D}|\times t)$ as $|\mathcal{Q}| \ll |\mathcal{D}|$. `InstRank` also has a near constant time cost because the distances calculated in previous feedback rounds are saved for the following rounds. `MRBIR` has to iteratively update the ranking result with expensive large matrix operations, resulting in the highest on-line retrieval time.

Although `ReFeat` has an off-line modeling stage, it costs only 2.87 s for the image database containing 10 000 images and 0.33 s for the music database containing 1000 songs, respectively. We believe that it pays to employ such an off-line modeling stage because of the good retrieval performance and quick processing time achieved by `ReFeat` for on-line retrieval.

Please cite this article as: G.-T. Zhou, et al., Relevance feature mapping for content-based multimedia information retrieval, Pattern Recognition (2011), doi:10.1016/j.patcog.2011.09.016

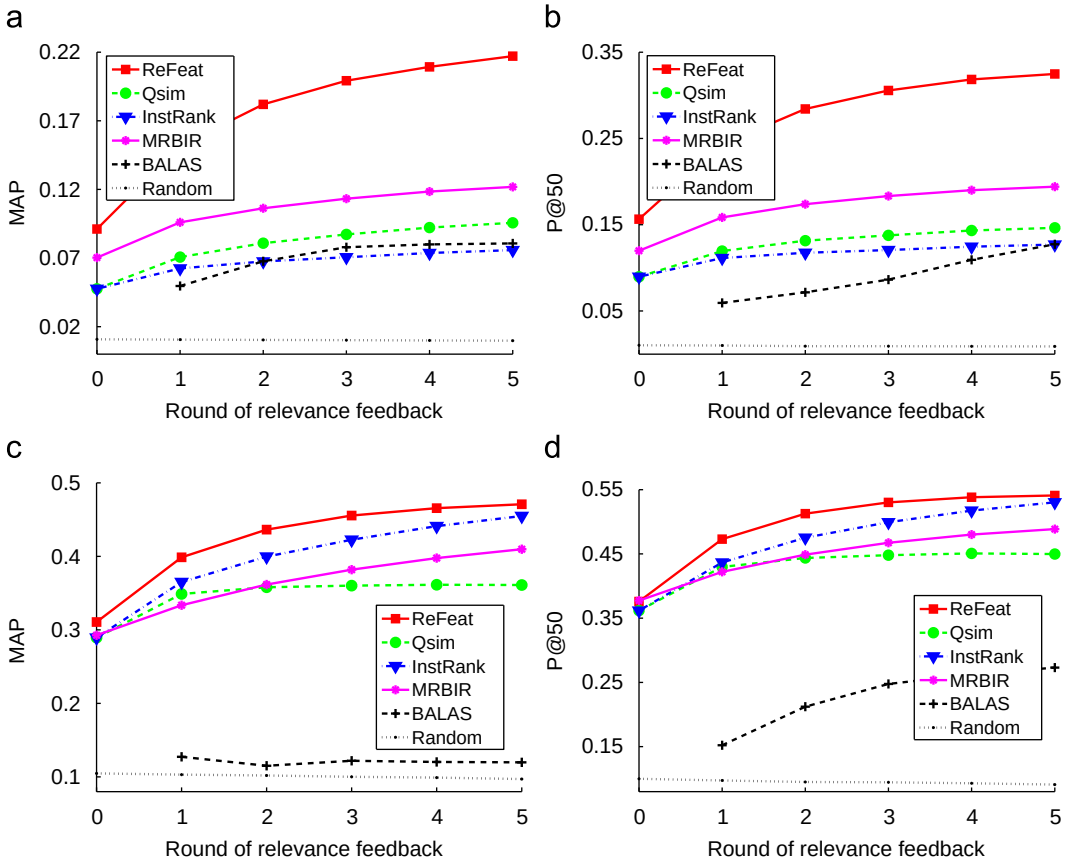


Fig. 4. Average MAP and P@50 values of ReFeat, Qsim, InstRank, MRBIR, BALAS and Random for retrieval in relevance feedback. (a) COREL image database: MAP. (b) COREL image database: P@50. (c) GTZAN music database: MAP. (d) GTZAN music database: P@50.

Table 4
Average on-line processing times (in millisecond) of ReFeat (RF), Qsim (QS), InstRank (IR), MRBIR (MR) and BALAS (BA).

Round	RF	QS	IR	MR	BA
(a) COREL image database					
0	27.2	**24.7**	**24.7**	612.9	N/A
1	**23.8**	71.3	32.6	1172.4	262.8
2	**24.0**	146.3	33.4	1172.3	317.5
3	**24.2**	261.9	34.2	1172.3	373.0
4	**24.4**	417.9	34.9	1172.2	437.9
5	**24.5**	615.8	35.5	1172.1	506.0
(b) GTZAN music database					
0	**3.6**	10.8	10.8	168.1	N/A
1	**3.1**	16.6	14.1	279.0	152.2
2	**3.3**	20.9	14.2	279.5	160.2
3	**3.4**	27.4	14.3	279.3	166.7
4	**3.6**	36.7	14.3	278.6	173.2
5	**3.7**	47.8	14.4	280.5	180.3

5.2. Analysis

This subsection analyzes some important issues in relation to ReFeat. We first empirically validate the effectiveness of the relevance feature mapping and our ranking scheme. Then we show the influence of increasing database dimension on the compared methods. At the end of this subsection, we study the effect of the three parameters in ReFeat and give some guidelines for selecting them. Note that the same conclusion can always be made for both MAP and P@50. Thus, we only provide the MAP results hereafter.

5.2.1. Relevance feature mapping

Recall that ReFeat is a two-stage process, where the first maps database instances to a relevance feature space, and the second ranks the instances in the new space. We conduct experiments to show the effectiveness of our relevance feature mapping in this subsection, and the efficacy of the proposed ranking scheme is validated in the next subsection.

Previous experiments have already shown that ReFeat outperforms existing methods which are conducted in the original feature space. Here, we hypothesize that the performance of existing methods can be improved using our relevance features. Thus, we perform three distance based methods, i.e., Qsim, InstRank and MRBIR, in our relevance feature space. The new methods are named Qsim-RF, InstRank-RF and MRBIR-RF, respectively. Table 5 presents the MAP results which are grouped in pairs for ease of comparison. Exactly the same relevance feature mapping is employed for all methods that use it. Note that round 0 gives the results with one query, and the Euclidean method performed in the original feature space is used as the base method for Qsim and InstRank. Similarly, Euclidean distance measured in the relevance feature space is employed by Qsim-RF and InstRank-RF at round 0.

As shown in Table 5, with the help of the relevance feature mapping, Qsim-RF, InstRank-RF and MRBIR-RF significantly outperform their original versions, i.e., Qsim, InstRank and MRBIR, respectively. There are two exceptions on the music database: the first is InstRank-RF which performs worse than InstRank, and the second is for retrieval with one query, Euclidean performs slightly better in the original space. Nevertheless, these observations show that our relevance feature space is more suitable for retrieval than the original space, and thus, we

Please cite this article as: G.-T. Zhou, et al., Relevance feature mapping for content-based multimedia information retrieval, Pattern Recognition (2011), doi:10.1016/j.patcog.2011.09.016

can conclude that the power of ReFeat is largely derived from the relevance feature mapping.

We also report the on-line processing time in Table 6. The time costs of Qsim-RF and InstRank-RF are expected to be longer than each of the original versions because the dimensionality of the relevance feature space is significantly higher than that of the original space. It is interesting to note that MRBIR-RF spends less time than MRBIR in most cases. This indicates that it is easier to find the underlying manifold in our relevance feature space, as compared to that in the original space.

Despite these improvements, ReFeat is still significantly better than the other three methods applied in the relevance feature space (except that MRBIR-RF achieves the best performance for retrieval with one query on the image database). The processing time reported in Table 6 also shows that ReFeat has the best efficiency among these methods. These results validate the efficacy and efficiency of our proposed ranking scheme. We will provide a more detailed analysis on our ranking scheme in the next subsection.

Table 5
Average MAP values ($\times 10^{-2}$) of ReFeat (RF), Qsim-RF (QS-RF), Qsim (QS), InstRank-RF (IR-RF), InstRank (IR), MRBIR-RF (MR-RF) and MRBIR (MR). The figures boldfaced are the best performance on each feedback round while the underlined indicate the better performance in each grouped pair.

Round	RF	QS-RF	QS	IR-RF	IR	MR-RF	MR
(a) COREL image database							
0	9.11	8.87	4.76	8.87	4.76	**10.88**	7.03
1	**15.17**	14.83	7.07	10.56	6.24	14.52	9.60
2	**18.20**	17.51	8.08	11.81	6.76	16.01	10.63
3	**19.92**	19.17	8.72	12.85	7.06	17.05	11.32
4	**20.93**	20.17	9.22	13.49	7.37	17.68	11.84
5	**21.71**	20.98	9.57	14.07	7.58	18.11	12.18
(b) GTZAN music database							
0	**31.07**	28.73	28.94	28.73	28.94	29.54	29.27
1	**39.87**	35.14	34.89	32.70	36.50	34.15	33.36
2	**43.64**	37.06	35.80	36.06	39.97	37.01	36.17
3	**45.56**	38.17	36.02	38.64	42.26	39.06	38.19
4	**46.56**	38.78	36.14	40.52	44.11	40.58	39.76
5	**47.09**	39.12	36.10	41.92	45.49	41.76	40.97

Table 6
Average on-line processing time (in millisecond) of ReFeat (RF), Qsim-RF (QS-RF), Qsim (QS), InstRank-RF (IR-RF), InstRank (IR), MRBIR-RF (MR-RF) and MRBIR (MR). The figures boldfaced are the smallest time on each feedback round while the underlined indicate the smaller time in each grouped pair.

Round	RF	QS-RF	QS	IR-RF	IR	MR-RF	MR
(a) COREL image database							
0	27.2	345.5	**24.7**	345.5	**24.7**	955.5	612.9
1	**23.8**	461.9	71.3	421.6	32.6	1117.4	1172.4
2	**24.0**	540.1	146.3	422.4	33.4	1117.5	1172.3
3	**24.2**	660.7	261.9	423.1	34.2	1117.5	1172.3
4	**24.4**	823.1	417.9	423.8	34.9	1117.5	1172.2
5	**24.5**	1030.2	615.8	424.5	35.5	1117.4	1172.1
(b) GTZAN music database							
0	**3.6**	34.2	10.8	34.2	10.8	96.6	168.1
1	**3.1**	45.6	16.6	42.2	14.1	114.8	279.0
2	**3.3**	51.0	20.9	42.3	14.2	114.9	279.5
3	**3.4**	59.7	27.4	42.4	14.3	114.9	279.3
4	**3.6**	71.2	36.7	42.5	14.3	114.8	278.6
5	**3.7**	86.4	47.8	42.5	14.4	114.8	280.5

5.2.2. The ranking scheme

This subsection analyzes our ranking scheme. ReFeat incorporates the query information into the feature weights. Here, we employ the same weights in the existing methods to improve their performance. Based on InstRank-RF, we design a new method called InstRank-WRF which uses weighted Euclidean distance instead of Euclidean distance in InstRank-RF. The weights for the relevance features are calculated in exactly the same way as that in ReFeat. InstRank-WRF is compared with ReFeat and InstRank-RF in Fig. 5 and Table 7. It is shown that InstRank-WRF outperforms InstRank-RF in most cases except for retrieval with one query on the image database. These observations show that the feature weights are not only useful in our ranking scheme, but also in existing distance-based ranking schemes. We also provide the retrieval performance of InstRank in Fig. 5. Recall that InstRank performs better than InstRank-RF on the music database. However, with the feature weights, InstRank-WRF is now better than InstRank.

Overall, Fig. 5 and Table 7 reveal that ReFeat is superior to InstRank-WRF in terms of both retrieval performance and

Table 7
Average on-line processing time (in millisecond) of ReFeat (RF), InstRank-WRF (IR-WRF) and InstRank-RF (IR-RF).

Round	COREL image database			GTZAN music database		
	RF	IR-WRF	IR-RF	RF	IR-WRF	IR-RF
0	**27.2**	731.6	345.5	**3.6**	98.5	34.2
1	**23.8**	1773.0	421.6	**3.1**	232.4	42.2
2	**24.0**	1881.2	422.4	**3.3**	237.9	42.3
3	**24.2**	1899.9	423.1	**3.4**	241.1	42.4
4	**24.4**	1920.9	423.8	**3.6**	244.1	42.5
5	**24.5**	1940.8	424.5	**3.7**	247.5	42.5

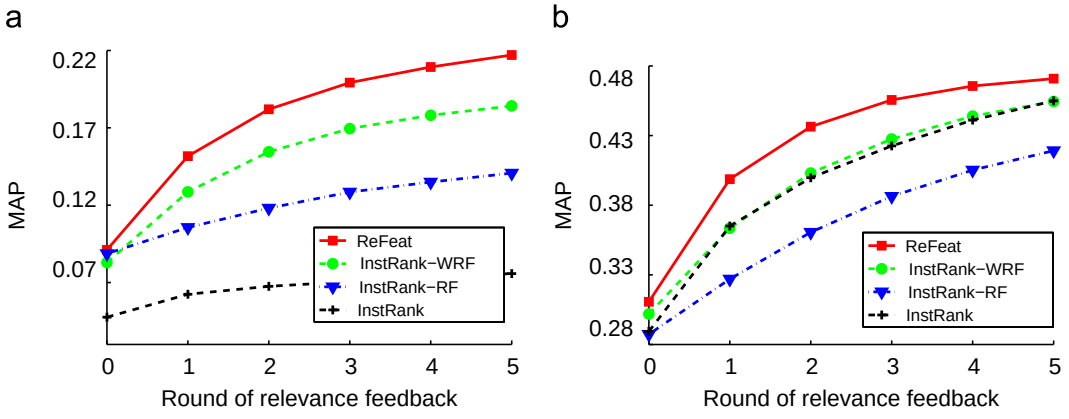


Fig. 5. Average MAP values of ReFeat, InstRank-WRF, InstRank-RF and InstRank. (a) COREL image database. (b) GTZAN music database.

Please cite this article as: G.-T. Zhou, et al., Relevance feature mapping for content-based multimedia information retrieval, Pattern Recognition (2011), doi:10.1016/j.patcog.2011.09.016

processing time. This indicates that there is no need to calculate the costly distance in the relevance feature space; instead, a good ranking can be efficiently produced by simply averaging the weighted relevance feature values.

5.2.3. Increasing dimensionality

Recall that every image in the COREL image database is represented by a 67-dimensional feature vector containing shape, texture and color features. Here we denote the database as $\text{COREL}_{[67]}$, and construct three other databases: (i) $\text{COREL}_{[11]}$ employs 11 shape features only; (ii) $\text{COREL}_{[35]}$ uses 35 features, consisting of 11 shape and 24 texture features; and (iii) $\text{COREL}_{[200]}$ is a 200-dimensional database, created by adding 133 random features to $\text{COREL}_{[67]}$ (each random feature is generated from a uniform distribution). Similarly, we denote the original GTZAN music database as $\text{GTZAN}_{[230]}$, and construct three other databases: (i) $\text{GTZAN}_{[20]}$ uses the first 20 features of $\text{GTZAN}_{[230]}$; (ii) $\text{GTZAN}_{[100]}$ employs the first 100 feature of $\text{GTZAN}_{[230]}$; and (iii) $\text{GTZAN}_{[400]}$ is created by adding 170 random features to $\text{GTZAN}_{[230]}$. All the methods are evaluated on the eight databases, and the retrieval results with one query and in feedback round 5 are shown in Fig. 6 and Table 8.

Fig. 6 shows that ReFeat outperforms the other methods regardless of how many features are used to describe the database. The only exception is $\text{GTZAN}_{[20]}$, on which Euclidean is slightly better than ReFeat for retrieval with one query. These results validate the efficacy of ReFeat when dealing with different dimensional databases.

Note that on the databases $\text{COREL}_{[200]}$ and $\text{GTZAN}_{[400]}$ with randomly generated features, ReFeat outperforms the other methods with the lowest performance degradation, when compared with that on the original databases $\text{COREL}_{[67]}$ and $\text{GTZAN}_{[230]}$, respectively. For example, at feedback round 5 of image retrieval, the MAP value of ReFeat degrades by 50.4%, which is much better than 93.7% for Qsim, 93.7% for InstRank, 94.7% for MRBIR and 92.4% for BALAS; at feedback round 5 of music retrieval, ReFeat only degrades by 11.3%, as compared to 56.7% for Qsim, 66.1% for InstRank, 84.5% for MRBIR and 51.3% for BALAS. These results show that ReFeat has a good tolerance to randomly generated or irrelevant features.

Moreover, it is interesting to note that for our music retrieval problem, every method (except MRBIR) achieves the best MAP on $\text{GTZAN}_{[100]}$ out of the four databases including the original one, $\text{GTZAN}_{[230]}$. This observation indicates that the music retrieval

Table 8
Average on-line processing time (in millisecond) of the methods tested on the image and music databases with different dimensions. The method names are abbreviated as ReFeat (RF), Euclidean (EU), MRBIR (MR), Qsim (QS), InstRank (IR) and BALAS (BA).

Database	One query			Round 5				
	RF	EU	MR	RF	QS	IR	MR	BA
(a) COREL image database								
$\text{COREL}_{[11]}$	27.2	**4.3**	591.6	24.5	590.9	**10.6**	1172.1	347.1
$\text{COREL}_{[35]}$	27.2	**13.7**	599.6	24.5	602.8	**22.5**	1172.1	415.7
$\text{COREL}_{[67]}$	27.2	**24.7**	612.9	**24.5**	615.8	35.5	1172.1	506.0
$\text{COREL}_{[200]}$	**27.2**	69.5	645.2	**24.5**	670.0	89.7	1172.1	874.2
(b) GTZAN music database								
$\text{GTZAN}_{[20]}$	3.7	**0.4**	144.7	3.7	35.0	**1.6**	280.5	24.8
$\text{GTZAN}_{[100]}$	3.7	**3.6**	147.1	**3.7**	37.9	4.5	280.5	50.0
$\text{GTZAN}_{[230]}$	**3.6**	10.8	168.1	**3.7**	47.8	14.4	280.5	180.3
$\text{GTZAN}_{[400]}$	**3.7**	13.8	159.0	**3.7**	50.6	17.2	280.5	148.5

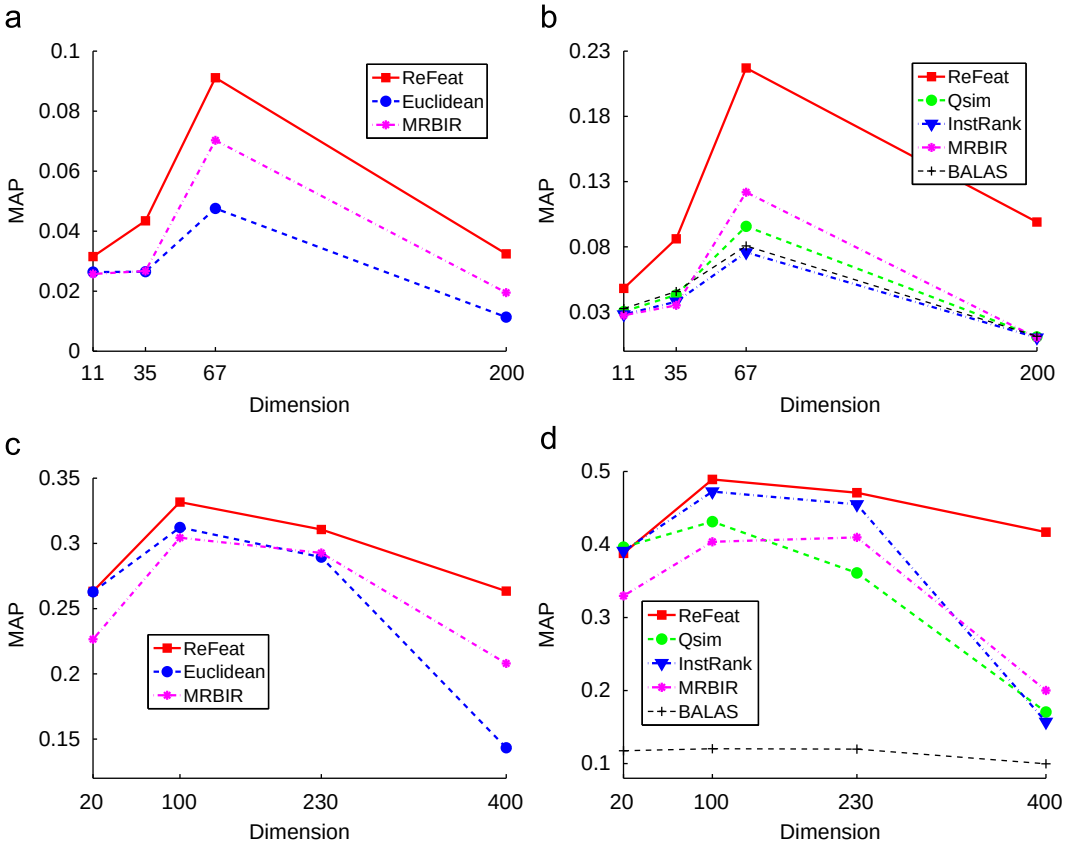


Fig. 6. Average MAP values of the compared methods evaluated on the image and music databases with different dimensions. (a) COREL image database: one query. (b) COREL image database: round 5. (c) GTZAN music database: one query. (d) GTZAN music database: round 5.

Please cite this article as: G.-T. Zhou, et al., Relevance feature mapping for content-based multimedia information retrieval, Pattern Recognition (2011), doi:10.1016/j.patcog.2011.09.016

performance might be further improved using some proper feature selection scheme.

The processing time reported in Table 8 shows that Euclidean and InstRank spend the shortest time in low-dimensional cases, but their processing time increases linearly with respect to the database dimension. Qsim, MRBIR and BALAS spend much more time than ReFeat. ReFeat achieves constant time with respect to the database dimension, either dealing with one query or handling feedbacks. This enables our framework to scale up to high-dimensional databases without increasing the processing time.

5.2.4. Using different numbers of relevance features

We study the effect of the number of relevance features, i.e., t, in this subsection. Fig. 7 shows the MAP values of ReFeat as t varies from 10, 50, 100, 500, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, to 10 000. Here we set $\psi=8$ and $\gamma=0.25$ by default.

Fig. 7 shows that the retrieval performance of ReFeat rapidly increases with the increase of t when t is relatively small. Even with a sufficiently large t, the performance still appears to rise without overfitting. These observations show the possibility of improving the performance of ReFeat by adding more relevance features. However, when setting t, the trade-off between performance and processing time should be considered.

5.2.5. Using different sub-sample sizes

From the analysis provided in Section 4.3, we know that iForests built with different sub-sample sizes generate different sets of topologically distinct iTrees, thus producing different sets of distinct path lengths. We suspect that the "diversity" of the path lengths has a critical impact on the performance of ReFeat, because a system with diverse path lengths tends to provide a full range of relevancy to improve ranking results. Therefore, to gain an insight into the setting of sub-sample size ψ, we use Shannon index [27] to measure the diversities of iForests built with different ψ values. Shannon index is a statistic for measuring the biodiversity of an ecosystem. The index increases when the ecosystem has additional unique species or a greater species evenness. A bigger Shannon index indicates a larger diversity. In this subsection, each instance (e.g., an image or a song) is treated as an ecosystem. The instance may have different relevance feature values on different iTrees, and each possible feature value is considered as a species in the ecosystem. We count the numbers of the species and measure the instance diversity by Shannon index. The final diversity of the iTrees is estimated by averaging the Shannon indices over all instances. Formally, the diversity $D(\psi)$ of the iTrees built with sub-sample size ψ is calculated by:

$$D(\psi)=-\frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}\sum_{j=1}^{|\mathcal{L}_\psi|}\left(\frac{n_j(\mathbf{x}_i)}{t}\ln\frac{n_j(\mathbf{x}_i)}{t}\right)-\frac{|\mathcal{L}_\psi|-1}{2t}, \tag{8}$$

where $\mathcal{L}_\psi=\{\ell_1,\ell_2,\ldots,\ell_k\}$ is the set of all possible relevance feature values measured by the iTrees, $\mathbf{x}_i\in\mathcal{D}$ is an instance in the database, $n_j(\mathbf{x}_i)$ returns the number of iTrees in which $\mathbf{x}_i$ has feature value ℓ_j, t is the total number of iTrees, and $(|\mathcal{L}_\psi|-1)/2t$ is a correction factor.

We set $\psi=2^2,2^3,\ldots,2^{12}$ for the image database, and $\psi=2^2,2^3,\ldots,2^9$ for the music database. The resultant Shannon indices are plotted in Fig. 8, which shows that the diversity increases as ψ increases from 4 and reaches the peak at $\psi=64$ on both the image and music databases. It is also interesting to note that the diversity decreases as ψ goes beyond 64, even though the number of possible feature values (i.e., possible species) increases. The MAP values of ReFeat are also shown in Fig. 8. Since the best performance of ReFeat is obtained with $\psi=8$ for the image database and $\psi=4$ for the music database, and there is no benefit to use a large ψ (i.e., $\psi>64$), we suspect that the optimal setting for any task is somewhere between the smallest ψ $(=4)$ and the diversity peak. This can be used as an empirical guideline for setting the sub-sample size.

5.2.6. Using different γ values

We also study how the trade-off parameter γ affects the performance of ReFeat in relevance feedback. We test it by varying γ from 0 to 1 with step 0.1, and the resultant MAP values are shown in Fig. 9. It shows that ReFeat achieves relative good performance when $\gamma\in[0.1,0.4]$ on both the image and music databases. These observations verify our statement in Section 4.4 that positive instances should contribute more than negative ones.

6. Discussion

This section discusses three related issues. We first provide some necessary characteristics of a ranking model to be applied in the ReFeat framework. Then, we detail the difference between our ranking scheme and that measured by distance and similarity. Finally, our ranking score calculation is compared with the one used in iForest for anomaly detection.

For a successful application in the proposed framework, the necessary characteristics of alternative ranking models are (i) each individual model provides a ranking of instances through some

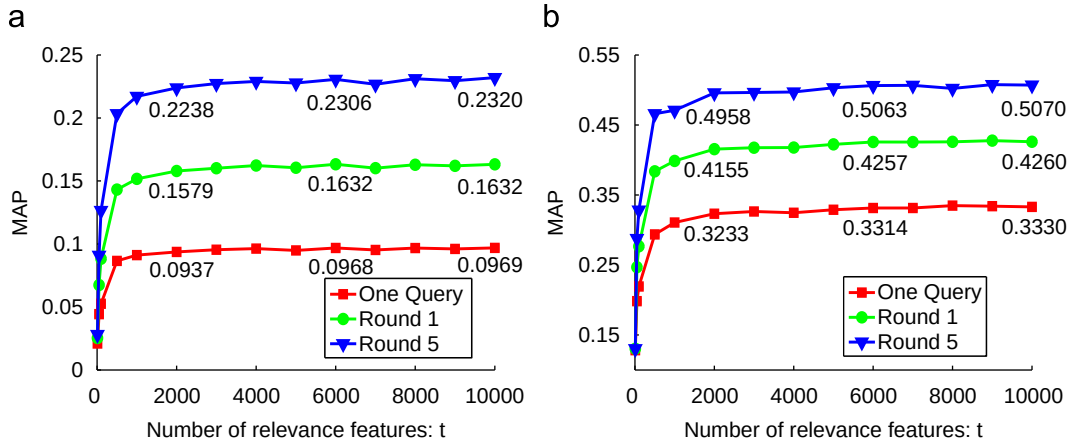


Fig. 7. Average MAP values of ReFeat with one query and in feedback rounds 1 and 5 using different number of relevance features. (a) COREL image database. (b) GTZAN music database.

Please cite this article as: G.-T. Zhou, et al., Relevance feature mapping for content-based multimedia information retrieval, Pattern Recognition (2011), doi:10.1016/j.patcog.2011.09.016

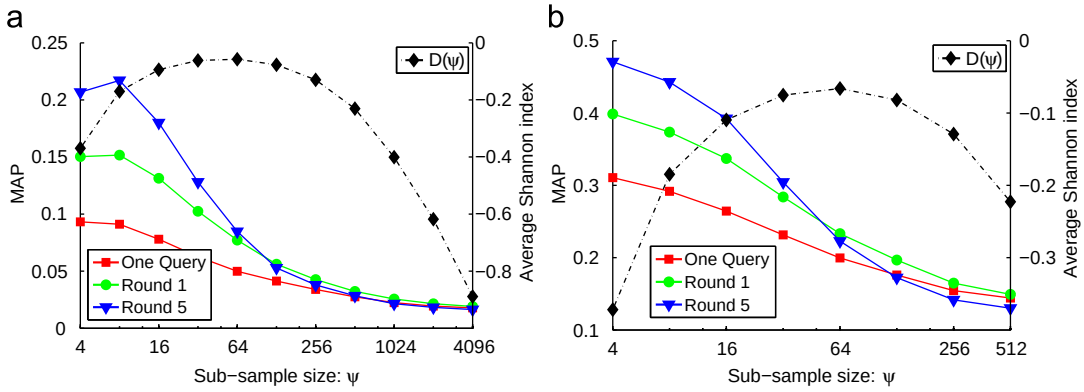


Fig. 8. Average MAP values of ReFeat with one query and in feedback rounds 1 and 5 using different sub-samples sizes. $D(\psi)$ is the average Shannon index calculated for the iTrees built with sub-sample size ψ. (a) COREL image database. (b) GTZAN music database.

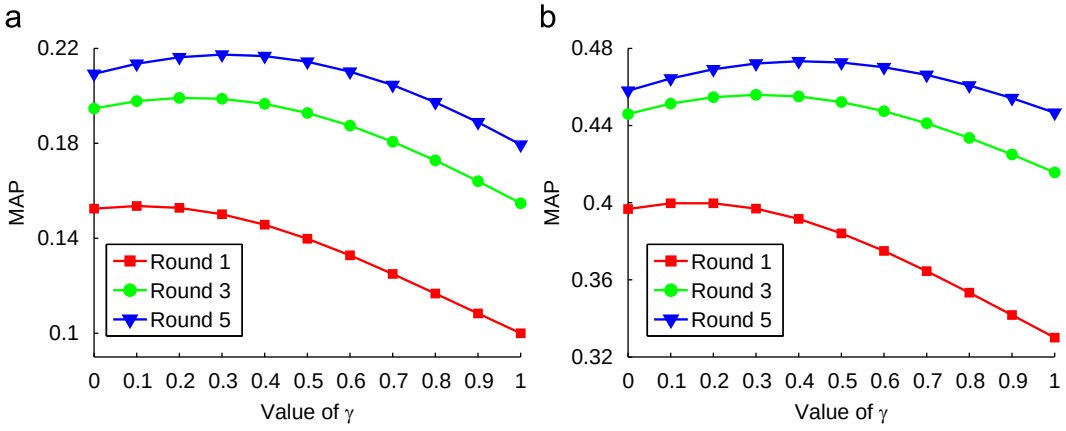


Fig. 9. Average MAP values of ReFeat in feedback rounds 1, 3 and 5 with different γ values. (a) COREL image database. (b) GTZAN music database.

profile underlying the database and (ii) each model is generated efficiently so that the multiple models, representing multiple profiles of the database, can be generated very quickly to form the relevance feature space. We show that iTrees work well in our framework. Whether there are other ranking models which satisfy the characteristics is an open question.

Next we analyze the difference between our ranking scheme and that measured by distance and similarity. Let $d(\mathbf{a},\mathbf{b})$ and $s(\mathbf{a},\mathbf{b})$ denote the distance value and similarity value, respectively, between two instances $\mathbf{a}$ and $\mathbf{b}$. Then a distance metric and its inversely related similarity measure are required to obey the following four axioms for all instances $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ [28]: (i) equal self-similarity: $d(\mathbf{a},\mathbf{a})=d(\mathbf{b},\mathbf{b})$ and $s(\mathbf{a},\mathbf{a})=s(\mathbf{b},\mathbf{b})$; (ii) minimality: $d(\mathbf{a},\mathbf{b})>d(\mathbf{a},\mathbf{a})$ and $s(\mathbf{a},\mathbf{b})<s(\mathbf{a},\mathbf{a})$; (iii) symmetry: $d(\mathbf{a},\mathbf{b})=d(\mathbf{b},\mathbf{a})$ and $s(\mathbf{a},\mathbf{b})=s(\mathbf{b},\mathbf{a})$; (iv) triangle inequality: $d(\mathbf{a},\mathbf{b})+d(\mathbf{b},\mathbf{c})>d(\mathbf{a},\mathbf{c})$, and if $\mathbf{a}$ and $\mathbf{b}$ are similar and $\mathbf{b}$ and $\mathbf{c}$ are similar, then $\mathbf{a}$ and $\mathbf{c}$ must also be similar.

The score calculated by Eq. (3) does not satisfy any of the above axioms. For example, symmetry does not hold in our calculation since $Score(\mathbf{a}|\mathbf{b})-Score(\mathbf{b}|\mathbf{a})=\sum_{i=1}^{t}(\ell_i(\mathbf{b})-\ell_i(\mathbf{a}))$, which is not 0 in most cases. The violation of the axioms provides our ranking scheme more flexibility when ranking instances with respect to a query. In fact, questions have been raised about the practical validity of each of these axioms [28]. To the best of our knowledge, there is no other CBMIR ranking scheme that violates all the axioms.

In the anomaly detection setting [23], instances are anomalies if they are irrelevant to the various profiles modeled by different iTrees, i.e., if they have short average path lengths in an iForest model. Thus, the anomaly scoring formulation given in [23] can be rewritten as $Score_{AD}(\mathbf{x})=\frac{1}{t}\sum_{i=1}^{t}\ell_i(\mathbf{x})$, where high scores indicate normal points, and low scores indicate anomalies. The above anomaly scoring formulation is only different from Eq. (3) by one term, which is the feature weight $w_i(\mathbf{q})$. We show that, under CBMIR, this term effectively modifies the ranking scheme from providing an ordering from normal points to anomalies under anomaly detection, to providing an ordering from instances most relevant to those most irrelevant with respect to the query $\mathbf{q}$.

7. Conclusions

This paper proposes a novel ranking framework for CBMIR with relevance feature mapping derived from an ensemble of ranking models. We employ an ensemble of iTrees to map instances from the original feature space to the proposed new relevance feature space. We show that the new relevance feature space has richer information than the original one for ranking database instances with respect to a given query as well as subsequent feedbacks. We also show that the relevance feature space accounts for the significant performance improvement of several existing methods when compared to the same methods applied in the original feature space. Moreover, our experiments validate the utility of our relevance feature weighting, on which the proposed new ranking scheme is based. The new scheme performs better than the four existing methods when they are evaluated in the same footing, in terms of both retrieval performance and time complexity.

The proposed framework has the following unique characteristics: (i) it utilizes no distance measure and has linear time and

Please cite this article as: G.-T. Zhou, et al., Relevance feature mapping for content-based multimedia information retrieval, Pattern Recognition (2011), doi:10.1016/j.patcog.2011.09.016

space complexities with respect to the database size when building its model and mapping the database off-line; (ii) it has constant on-line retrieval time, irrespective of the number of relevance feedback rounds; (iii) it can deal with high-dimensional databases with constant time complexity, once the number of relevance features is fixed; and (iv) it has a good tolerance to irrelevant features.

Acknowledgements

This work is supported in part by the National Natural Science Foundation of China under No. 61070097 and the Research Fund for the Doctoral Program of Higher Education under Grant No. 20100131110021.

We are grateful for the anonymous reviewers for their suggestions and comments, which have led to significant improvements of this paper. Zhi-Hua Zhou had given Guang-Tong a strong foundation in CBMIR when Guang-Tong visited Nanjing University for six months prior to this project. We would also like to thank Zhouyu Fu for his many helpful discussions and technical assistance with regard to the music data set.

Appendix A. Isolation forest

This section briefly introduces the methodology of iForest [23], which employs a two-stage process to detect anomalies. We provide some insights on how each iTree measures the relevance of instances with respect to a profile underlying the data. It helps to understand the relevance feature space.

In the first stage, iForest builds a collection of iTrees using fixed-sized random sub-samples of a data set. Each iTree is constructed by recursively random-partitioning the sub-sample along axis-parallel coordinates until every instance is isolated from the rest of instances or a specified height limit is reached. The algorithmic details are given by Algorithms 1 and 2. Note that an iTree models a profile of the given random sub-sample, and different iTrees describe different profiles due to the randomness incurred in both the sub-sampling process and the tree building process.

Algorithm 1. *iForest*($\mathcal{D}$,t,ψ).

input : $\mathcal{D}$ - input data, t - number of iTrees, ψ - sub-sample size
output: a set of t iTrees
set height limit $h=\lceil \log_2(\psi) \rceil$;
for $i=1$ *to* t **do**
  $D \leftarrow sample(\mathcal{D},\psi)$; // randomly sample ψ instances from $\mathcal{D}$
  $T_i \leftarrow iTree(D,0,h)$;
end

Algorithm 2. *iTree*(D,e,h).

input : D - input data, e - current tree height, h - height limit
output: an iTree
if $e \geq h$ *or* $|D| \leq 1$ **then**
  return *exNode*{*Size* $\leftarrow |D|$}; // an external node
else
  randomly select an attribute a from the data D;
  randomly select a split point p from *max* and *min* values of attribute a in D;
  $D_l \leftarrow filter(D,a<p)$; // instances in D which have values less than p on attribute a
  $D_r \leftarrow filter(D,a \geq p)$; // instances in D which have values greater than or equal to p on attribute a
  return *inNode*{*SplitAtt* $\leftarrow a$, *SplitValue* $\leftarrow p$, *Left* $\leftarrow iTree(D_l,e+1,h)$, *Right* $\leftarrow iTree(D_r,e+1,h)$}; // an internal node
end

In the second stage, iForest calculates an anomaly score for each test instance based on its average path length over all iTrees. A path length is estimated by counting the number of edges from the root node to the external node as an instance travels through the iTree. If the instance falls into an external node with $Size>1$, the returned path length is adjusted by adding $c(Size)$, which is defined in Eq. (2) and accounts for the average path length of an unbuilt subtree beyond the height limit. This process is given by Algorithm 3.

Algorithm 3. *PathLength*($\mathbf{x}$,T,e).

input : $\mathbf{x}$ - an instance, T - an iTree, e - current path length (to be initialized to 0 when first called)
output: the path length of $\mathbf{x}$
if *T is an external node* **then**
  return $e+c(T.Size)$; // $c(\cdot)$ is defined in Eq. (2)
end
$a \leftarrow T.SplitAtt$, $p \leftarrow T.SplitValue$;
if $\mathbf{x}_a < p$ **then**
  return *PathLength*($\mathbf{x}$,$T.Left$,$e+1$);
else
  return *PathLength*($\mathbf{x}$,$T.Right$,$e+1$);
end

Here, a short path length means that we can easily isolate the instance from the majority of instances by a few random partitions. Thus, instances having short path lengths always differ from the majorities on some characteristics. Note that an iTree describes a data profile from a given sub-sample. Therefore, instances having short path length have different data characteristics to the majorities which have long path lengths. Thus, the path length stipulated by an iTree actually measures the relevance of an instance with respect to the profile modeled by this iTree: a short (long) path length indicates that the instance is irrelevant (relevant) to the profile. For anomaly detection, instances identified to be irrelevant to the various profiles modeled by a number of iTrees are deemed to be anomalies, and instances relevant to the various profiles are normal points.

References

[1] M.S. Lew, N. Sebe, C. Djeraba, R. Jain, Content-based multimedia information retrieval: state of the art and challenges, ACM Transactions on Multimedia Computing, Communications, and Applications 2 (1) (2006) 1–19.
[2] R. Zhang, Z.M. Zhang, BALAS: empirical Bayesian learning in the relevance feedback for image retrieval, Image and Vision Computing 24 (3) (2006) 211–223.
[3] A.W.M. Smeulders, M. Worring, S. Santini, A. Gupta, R. Jain, Content-based image retrieval at the end of the early years, IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (12) (2000) 1349–1380.
[4] R. Datta, D. Joshi, J. Li, J.Z. Wang, Image retrieval: ideas, influences and trends of the new age, ACM Computing Surveys 40 (2) (2008) 1–60 (Article 5).
[5] R. Typke, F. Wiering, R.C. Veltkamp, A survey of music information retrieval systems, in: Proceedings of the Sixth International Conference on Music Information Retrieval, London, UK, 2005, pp. 153–160.
[6] C. Weihs, U. Ligges, F. Mörchen, D. Müllensiefen, Classification in music research, Advances in Data Analysis and Classification 1 (3) (2007) 255–291.
[7] Y. Rui, T.S. Huang, M. Ortega, S. Mehrotra, Relevance feedback: a power tool for interactive content-based image retrieval, IEEE Transactions on Circuits and Systems for Video Technology 8 (5) (1998) 644–655.
[8] X.S. Zhou, T.S. Huang, Relevance feedback in image retrieval: a comprehensive review, Multimedia Systems 8 (6) (2003) 536–544.
[9] J. He, M. Li, H. Zhang, H. Tong, C. Zhang, Manifold-ranking based image retrieval, in: Proceedings of the Twentieth ACM International Conference on Multimedia, New York, 2004, pp. 9–16.

Please cite this article as: G.-T. Zhou, et al., Relevance feature mapping for content-based multimedia information retrieval, Pattern Recognition (2011), doi:10.1016/j.patcog.2011.09.016

[10] G. Giacinto, F. Roli, Instance-based relevance feedback for image retrieval, in: Advances in Neural Information Processing Systems, vol. 17, Vancouver, Canada, 2005, pp. 489–496.
[11] Z.-H. Zhou, H.-B. Dai, Query-sensitive similarity measure for content-based image retrieval, in: Proceedings of the Sixth IEEE International Conference on Data Mining, Hong Kong, China, 2006, pp. 1211–1215.
[12] D. Zhou, J. Weston, A. Gretton, O. Bousquet, B. Schölkopf, Ranking on data manifolds, in: Advances in Neural Information Processing Systems, vol. 16, Vancouver, Canada, 2003, pp. 169–176.
[13] A. Frome, Y. Singer, F. Sha, J. Malik, Learning globally consistent local distance functions for shape-based image retrieval and classification, in: Proceedings of the Eleventh International Conference on Computer Vision, Rio de Janeiro, Brazil, 2007, pp. 1–8.
[14] J.-E. Lee, R. Jin, A. K. Jain, Rank-based distance metric learning: an application to image retrieval, in: Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition, Anchorage, AK, 2008, pp. 1–8.
[15] G. Wu, E. Y. Chang, N. Panda, Formulating context-dependent similarity functions, in: Proceedings of the Thirteenth ACM International Conference on Multimedia, Singapore, 2005, pp. 725–734.
[16] X. He, W.-Y. Ma, H. Zhang, Learning an image manifold for retrieval, in: Proceedings of the Twentieth ACM International Conference on Multimedia, New York, 2004, pp. 17–23.
[17] Y.-Y. Lin, T.-L. Liu, H.-T. Chen, Semantic manifold learning for image retrieval, in: Proceedings of the Thirteenth ACM International Conference on Multimedia, Singapore, 2005, pp. 249–258.
[18] Y. Rui, T.S. Huang, S. Mehrotra, Content-based image retrieval with relevance feedback in MARS, in: Proceedings the 1997 International Conference on Image Processing, Washington, DC, 1997, pp. 815–818.
[19] N. Panda, E.Y. Chang, Efficient top-k hyperplane query processing for multimedia information retrieval, in: Proceedings of the Fourteenth ACM International Conference on Multimedia, Santa Barbara, CA, 2006, pp. 317–326.
[20] J.A. Aslam, M.H. Montague, Models for metasearch, in: Proceedings of the Twenty-Fourth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New Orleans, LA, 2001, pp. 275–284.
[21] N. Rasiwasia, P.J. Moreno, N. Vasconcelos, Bridging the gap: query by semantic example, IEEE Transactions on Multimedia 9 (5) (2007) 923–938.
[22] R. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval, Addison Wesley Longman, Boston, MA, 1999.
[23] F.T. Liu, K.M. Ting, Z.-H. Zhou, Isolation forest, in: Proceedings of the Eighth IEEE International Conference on Data Mining, Pisa, Italy, 2008, pp. 413–422, software download at 〈http://sourceforge.net/projects/iforest/〉.
[24] Z.-H. Zhou, K.-J. Chen, H.-B. Dai, Enhancing relevance feedback in image retrieval using unlabeled data, ACM Transactions on Information Systems 24 (2) (2006) 219–244.
[25] G. Tzanetakis, P.R. Cook, Musical genre classification of audio signals, IEEE Transactions on Speech and Audio Processing 10 (5) (2002) 293–302.
[26] M.I. Mandel, D. Ellis, Song-level features and support vector machines for music classification, in: Proceedings of the Sixth International Conference on Music Information Retrieval, London, UK, 2005, pp. 594–599.
[27] C.J. Krebs, Ecological Methodology, HarperCollins, New York, 1989.
[28] S. Santini, R. Jain, Similarity measures, IEEE Transactions on Pattern Analysis and Machine Intelligence 21 (9) (1999) 871–883.

Guang-Tong Zhou received his B.Sc. and M.Sc. degrees from Shandong University in 2007 and 2010, respectively. He is currently a Ph.D. candidate at School of Computing Science, Simon Fraser University. His research interests include data mining, machine learning and their applications to content-based image retrieval, fingerprint recognition and social network analysis.

Kai Ming Ting received his Ph.D. from the University of Sydney, Australia. Later, he worked at the University of Waikato, New Zealand and Deakin University, Australia. He joined Monash University since 2001 and currently serves as the Associate Dean Research Training in Faculty of Information Technology and an Associate Professor in Gippsland School of Information Technology at Monash University. He had previously held visiting positions at Osaka University, Japan, Nanjing University, China, and Chinese University of Hong Kong.

His current research interests are in the areas of mass estimation and mass-based approaches, ensemble approaches, data stream data mining, and swarm intelligence. He is an associate editor for Journal of Data Mining and Knowledge Discovery. He had co-chaired the Pacific-Asia Conference on Knowledge Discovery and Data Mining 2008 and will co-chaired the Pacific Rim International Conference on Artificial Intelligence 2012. He had served as a member of program committees for a number of international conferences including ACM SIGKDD, IEEE ICDM, and ICML. His research projects are supported by grants from Australian Research Council, US Air Force of Scientific Research (AFOSR/AOARD), and Australian Institute of Sport.

Fei Tony Liu received his Ph.D. in 2011 from Monash University. During his post-graduate studies, he was awarded with the Best Paper and Best Student Paper Awards in the Tenth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2006) and the Runner-Up Best Theoretical/Algorithms Paper Award in the IEEE International Conference on Data Mining (ICDM 2008). His research interests include ensemble learning, outlier detection, and predictive classification.

Yilong Yin is the Director of MLA Group and a Professor of Shandong University. He received his Ph.D. degree in 2000 from Jilin University. From 2000 to 2002, he worked as a post-doctoral fellow in the Department of Electronic Science and Engineering, Nanjing University. His research interests include machine learning, data mining, and biometrics.

Please cite this article as: G.-T. Zhou, et al., Relevance feature mapping for content-based multimedia information retrieval, Pattern Recognition (2011), doi:10.1016/j.patcog.2011.09.016

MassCfier: A new generative classifier based on Mass

Sunil Aryal
Gippsland School of IT
Monash University, Australia
sunil.aryal@monash.edu

Kai Ming Ting
Gippsland School of IT
Monash University, Australia
kaiming.ting@monash.edu

Jonathan R. Wells
Gippsland School of IT
Monash University, Australia
jonathan.wells@monash.edu

ABSTRACT

The current approach to build generative classifiers is to estimate the joint probability $p(\mathbf{x}, y)$ indirectly by estimating the conditional likelihood $p(\mathbf{x}|y)$ and the prior probability $p(y)$. They predict the most likely class that maximises the posterior probability $p(y|\mathbf{x})$ using Bayes rule.

In this research, we propose a new type of generative classifier called **MassCfier**, that estimates the joint distribution directly through a recently introduced data modelling mechanism called mass estimation. This new generative classifier makes prediction based on a new decision rule that maximises mass, rather than Bayes rule, and has sub-linear time complexity and constant space complexity.

Empirical evaluations show that MassCfier performs better than or at least competitive with existing generative classifiers based on Bayes rule on benchmark data sets in terms of predictive accuracy.

Categories and Subject Descriptors

[**Algorithms**]: Classification

Keywords

Mass distribution, Mass Estimation, Generative classifier, MassCfier

1. INTRODUCTION

Classification is a data mining task that deals with assigning data instances described by a set of variables ($\mathbf{x}$) to one of the predefined mutually exclusive categories (y).

Discriminative and generative classifiers are two distinct approaches to solve classification problems [10, 13]. Generative classifiers model the joint probability $p(\mathbf{x}, y)$ via Bayes rule [10]. Discriminative classifiers, on the other hand, learn a direct mapping from $\mathbf{x}$ to y [10]. Classifers such as Naive Bayes (NB), Bayesian Belief Network (BayesNet), Aggregating One Dependence Estimators (AODE) are examples of generative classifiers; whereas, Artificial Neural Networks (ANN), Linear Logistic Regression (LLR), Support Vector Machines (SVM) are examples of discriminative classifiers.

Building generative models require density estimators. Current density estimators such as kernel density estimator and k-nearest neighbour density estimator have a high time and space complexities. Thus, it is difficult to estimate $p(\mathbf{x}, y)$ directly to build generative models even with data sets that have a moderate number of dimensions and moderate data size.

Instead, the current generative approach focuses on estimating $p(\mathbf{x}|y)$ and $p(y)$, and makes the final decision via Bayes rule. This approach encounters the same limitation of existing density estimators: $p(\mathbf{x}|y)$ cannot be estimated directly. However, surrogates of $p(\mathbf{x}|y)$ can be estimated efficiently provided some assumptions are made (e.g., attribute independence given the class.) Though this type of generative classifiers has been shown to perform well [7, 12, 8], the assumptions made are often violated in practice and can result in poor predictive accuracy.

Mass estimation [17, 16, 15] provides an alternative to density estimation for data modelling and it has been shown to work well in anomaly detection, information retrieval, clustering and regression. This paper is motivated to employ mass estimation to solve classification problems, in particular, by estimating joint distribution directly to build generative models. This is a more direct approach than the current approach to build generative models.

We propose a new type of generative classifier called **MassCfier** that exploits the notion of mass and mass distribution to estimate the joint distribution effectively.

MassCfier has three distinctive characteristics compared to existing generative classifiers:

1. The joint distribution is estimated directly without estimating the likelihood $p(\mathbf{x}|y)$ and the prior probability $p(y)$.
2. Its prediction decision is based on a maximum mass rule rather than Bayes rule.
3. It has sub-linear time complexity and constant space complexity; therefore, it scales better for very large databases.

The rest of the paper is structured as follows. We briefly discuss three existing generative classifiers in section 2 followed by the concept of mass and mass estimation in section 3. The proposed classifier, MassCfier, is described in section 4. The empirical evaluation results are presented in section 5. Finally, we provide discussion and conclusion in the last two sections.

2. EXISTING GENERATIVE CLASSIFIERS

The existing generative classifiers estimate the conditional likelihood $p(\mathbf{x}|y)$ and the prior $p(y)$ and use Bayes rule to make the final prediction.

$$\hat{y} = \arg\max_{y}(p(\mathbf{x}|y) \times p(y)) \quad (1)$$

Different generative classifiers estimate the conditional likelihood $p(\mathbf{x}|y)$ in different ways. We briefly describe three existing generative classifiers in this section.

2.1 Naive Bayes

Naive Bayes (**NB**) [2, 7] assumes class conditional independence and estimates the likelihood on each dimension separately. A typical structure of Naive Bayes is given in figure 1. The likelihood of $\mathbf{x}$ given class y is estimated as follows.

$$p(\mathbf{x}|y) = \prod_{i=1}^{d} p(x_i|y) \quad (2)$$

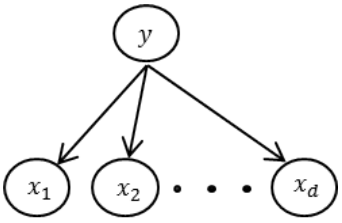


Figure 1: Structure of Naive Bayesian classifier where predictive attributes $(x_1, x_2, \cdots, x_d)$ are conditionally independent given the class attribute y.

For continuous valued attributes, $p(x_i|y)$ can be computed either through discretisation or by using a density estimator. We use three versions of Naive Bayes: (i) Naive Bayes with Gaussian Distribution (**NB-GD**) [7] (by assuming Gaussian distribution); (ii) Naive Bayes with Kernel Density Estimation (**NB-KDE**) [8]; (iii) Naive Bayes with Discretisation (**NB-Disc**).

2.2 Bayesian Networks

Bayesian Networks (**BayesNet**) [5, 11] learns probabilistic relationships among the attributes including the class in the form of directed acyclic graph (DAG) from the training data. In a graph, edges represent conditional dependencies and nodes, which are not connected, are conditionally independent. At each node, conditional probabilities with respect to its parents are learned from the training data. Figure 2 shows an example of a Bayesian Network representing a probabilistic relationship between four attributes and a class label. In a graph, each node are independent of its non-descendants given the state of its parents. In this way, the number of parameters needed to characterise a probability distribution are reduced and can be computed efficiently.
In the version of Bayesian Networks we used, continuous valued attributes are discretised.

2.3 Aggregating One-Dependence Estimator

Aggregating One-Dependence Estimators (**AODE**) [18] allows conditional dependence with one 'privileged' attribute.

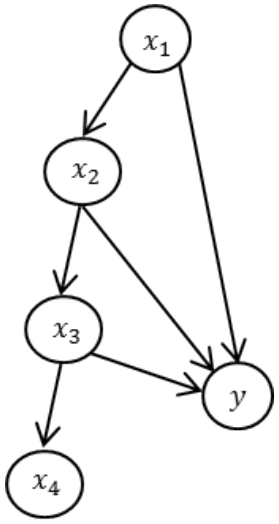


Figure 2: Bayesian Network representing a probabilistic relationship between four predictive attributes (x_1, x_2, x_3, x_4) and class variable y.

Other attributes are conditionally independent given class label y. The conditional probability, with a privileged attribute x_i, is computed as follows.

$$p(\mathbf{x}|x_i, y) = \prod_{j=1}^{d} p(x_j|x_i, y) \quad (3)$$

As AODE is designed for discrete attributes, continuous-valued attributes are discretised. The conditional probability is computed as relative frequencies as in NB-Disc.
Each attribute gets a chance to be a privilege attribute once; hence, AODE builds d models and aggregates the decisions to make the final prediction. Figure 3 shows a typical structure of AODE.

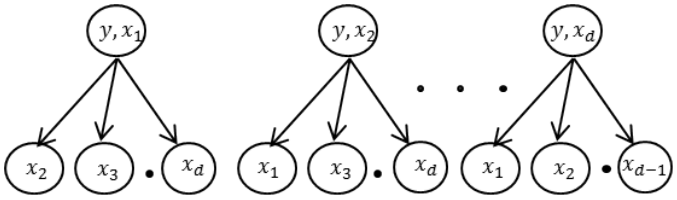


Figure 3: Typical structure of Aggregating One-Dependence Estimators (AODE).

3. MASS AND MASS-ESTIMATION

Ting et al [17] introduced the fundamental concept of mass as a base measure. The application of mass to solve various data mining problems such as regression, information retrieval, clustering, anomaly detection, and data stream are demonstrated in [17, 16, 14, 15]. Mass-based data mining methods often performed better than or at least as well as the state-of-the-art methods. The key advantages of mass-based methods are as follows:

1. Employ no distance measures and generally run faster.
2. Have average case sub linear time complexity and constant space complexity; hence, it can be applied to very large data sets.

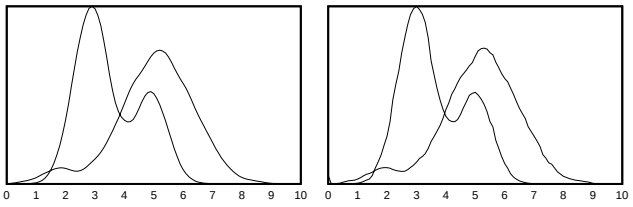
In its simplest form, mass is the number of data instances in a bounded region. A mass base function is defined as follows [16]:

$$\mathbf{m}(T(\mathbf{x})) = \begin{cases} \mathsf{m} & \text{if } \mathbf{x} \text{ is in a region of } T(\cdot), \\ 0 & \text{otherwise,} \end{cases} \quad (4)$$

where, $T(\cdot)$ is a function that subdivides the feature space of the given data set D into non-overlapping regions; and, m is the number of instances in a region of $T(\mathbf{x})$ in which $\mathbf{x}$ falls into.
Ting and Wells [16] showed that mass can also be effectively estimated by using data subsets $\mathcal{D}_i \subset D$ $(i = 1, \ldots, t)$, where $|\mathcal{D}_i| = \psi \ll n$. Each $\mathcal{D}_i$ is sampled without replacement from D and is used to construct $T_i(.)$. The estimated mass for an instance $\mathbf{x}$ is defined [16] as:

$$\overline{mass}(\mathbf{x}) = \frac{1}{t}\sum_{i=1}^{t} \mathbf{m}(T_i(\mathbf{x})) \quad (5)$$

Mass estimation has been shown to be a good data modelling mechanism in [17, 16, 15]. Figure 4 shows the estimation of two overlapped clusters in one dimensional feature space using kernel density estimation (KDE) and mass estimation. It demonstrates that mass-based estimation is comparable to that of KDE.



(a) Density distribution (b) Mass distribution

Figure 4: Density estimation of KDE vs. Mass estimation. Y-axis: a) density, b) mass. The parameters used for mass estimation are $t = 100$, $\psi = 4096$ and $h = 5$. These parameters are discussed in the following section. Parameter h for mass is equivalent to the bandwidth smoothing parameter for KDE. The bandwidth parameter was automatically selected in the case of KDE.

Once the data distribution has been modelled using mass distribution, a simple decision rule based on maximum mass can be used to make a prediction in the classification context.

4. PROPOSED CLASSIFIER

MassCfier is a generative classifier that exploits the notion of mass and mass distribution. It estimates the mass joint distribution of $\mathbf{x}$ and y. The corresponding mass base function $\mathbf{m}(T(\mathbf{x}), y)$ is defined as the count of instances in a region of $T(\mathbf{x})$ that belong to class y.

$$\mathbf{m}(T(\mathbf{x}), y) = \begin{cases} \mathsf{m}_y & \text{if } \mathbf{x} \text{ is in a region of } T(\cdot), \\ 0 & \text{otherwise,} \end{cases} \quad (6)$$

where, m_y is the number of instances belonging to class y in a region of $T(\mathbf{x})$.

The joint mass distribution of $\mathbf{x}$ and y is estimated as:

$$\overline{mass}(\mathbf{x}, y) = \frac{1}{t}\sum_{i=1}^{t} \mathbf{m}(T_i(\mathbf{x}), y) \quad (7)$$

where, each $T_i(.)$ is constructed by using data subsets $\mathcal{D}_i \subset D$ $(i = 1, \ldots, t)$.
At **training phase**, only $T_i(.)$ $(i = 1, \ldots, t)$ is to be constructed.
To predict a class for an instance $\mathbf{x}$, at **testing phase**, the joint mass distribution $\overline{mass}(\mathbf{x}, y)$ is used to make the final decision based on the following decision rule:

$$\hat{y} = \arg\max_y(\overline{mass}(\mathbf{x}, y)) \quad (8)$$

Note that $\overline{mass}(\mathbf{x}, y)$ is proportional to the joint probability $p(\mathbf{x}, y)$.
The decision rules of existing generative classifiers and MassCfier are provided in Table 1.

Table 1: Decision rules of different existing generative classifiers and MassCfier.

Classifier	Decision Rule	Remarks
NB-GD	$\arg\max_y(p(y) \times p(\mathbf{x}\|y))$	$p(x_i\|y)$ in eq.2 is estimated with norm. dist.
NB-KDE		$p(x_i\|y)$ in eq.2 is estimated with KDE.
NB -Disc		$p(x_i\|y)$ in eq.2 is estimated by discretisation.
Bayes Net	$\arg\max_y(\prod_i^d p(\pi_i, y)p(x_i\|\pi_i, y))$	$\pi_i = parents(x_i)$
AODE	$\arg\max_y(\sum_{i=1}^d p(x_i, y)p(\mathbf{x}\|x_i, y))$	$p(\mathbf{x}\|x_i, y)$ is estimated using eqn. 3
MassCfier	$\arg\max_y(\overline{mass}(\mathbf{x}, y))$	$\overline{mass}(\mathbf{x}, y)$ is estimated using eqn. 7

4.1 Implementation

Mass estimation can be implemented in different ways [17, 16, 14]. We use the implementation described in [16]. Feature space is divided into small non-overlapping regions by recursively splitting each dimension at mid point of the sample range.
$T(\cdot)$ is represented as a binary tree (called h:d-`Tree` in [16]). A parameter h defines the maximum level of binary subdivision. The maximum height of a tree is $h \times d$. Let Δ be a work space in $\mathcal{R}^d$ which envelops $\mathcal{D}$; and Δ has its length along each dimension j as $\Delta_j = \max(x_{kj}|\mathbf{x}_k \in \mathcal{D}) - \min(x_{kj}|\mathbf{x}_k \in \mathcal{D})$. The work space Δ is adjusted to become δ using a random perturbation conducted as follows. For each dimension j, a split point v_j is chosen randomly within the range Δ_j. Then, the new range δ_j along dimension j is defined as $[v_j - r, v_j + r]$, where $r = max(v_j - min_j(\Delta), max_j(\Delta) - v_j)$. The new ranges on all dimensions define the adjusted work space for the tree building process.
A subset $\mathcal{D}$ is constructed from D by sampling ψ instances

from each class without replacement. If there are not enough instances to sample in a class, all the instances from that class are used i.e. $\psi_y = min(\psi, |C_y|)$, where C_y is the set of instances belonging to class y and $|\mathcal{D}_i| = \sum_{y=1}^{c}(\psi_y)$. If every class in a data set has instances less than ψ, the entire training set D is used to build t trees. The random adjustment of the work space, as described earlier, ensures that no two trees are likley to be identical, even if the same data sub-sample is used to generate the trees.

The dimension to split is selected from a randomised set of d dimensions in a round-robin manner at each level of a tree. A tree is constructed by splitting the work space into two equal-volume half spaces at each level. The process is then repeated recursively on each non-empty half-space. The tree building process stops when there are less than two instances in a node or the maximum height is reached.

Figure 5 shows a typical example of h:d-`Tree` for $h = 1$ and $d = 2$. The dotted lines enclosed the instances in $\mathcal{D}$ (having 8 samples from each of the two classes) and the solid lines enclosed the adjusted work space. The algorithm, used to

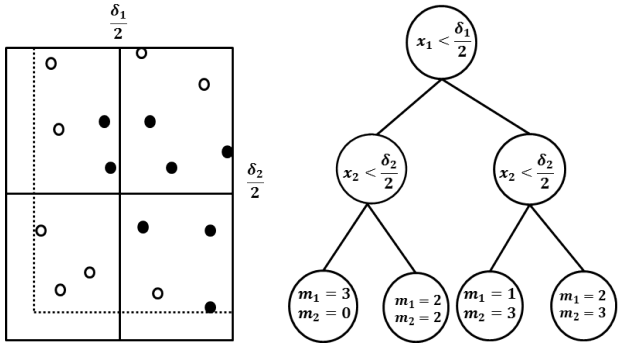


Figure 5: An example of h:d-`Tree` for $h = 1$ and $d = 2$.

generate such trees, is given in Appendix B for ease of reference.

The time complexity of constructing the trees is $O(tc\psi hd)$. The space complexity is $O(thd + n)$ during construction. After the trees are built, the data set is discarded, yielding $O(thd)$.

5. EXPERIMENTS

In this section, we compare the performance of MassCfier with existing generative classifiers Naive Bayes with density estimation through Gaussian Distribution assumption (NB-GD), Naive Bayes with Kernel Density Estimator (NB-KDE), Naive Bayes with Discretisation (NB-Disc), Bayesian Networks (BayesNet), and Aggregating One-dependence Estimators (AODE).

We have implemented the proposed method using the WEKA platform [6, 19] which has all of the existing generative classifiers. The data sets used are from UCI Machine Learning Repository [4] unless stated otherwise.

All the experiments were conducted as single thread jobs processed at 2.27 GHz on a Linux cluster using a node with 40 GB memory.

All the algorithms were executed with default parameters except BayesNet. For BayesNet, the parameter 'max number of parents' was set to 100 to enforce no restriction on the number of parents that a node can have in the network; and the parameter 'initialise as Naive Bayes' was set to 'false' to initialise an empty network structure. The rest of the parameters were set to defaults. The default settings for MassCfier were $t = 100$, $\psi = 4096$ and $h = \lceil \log_2(\psi) \rceil$. Where there are less than 4096 instances in each class, the entire data set were used to construct the trees.

We report the results in terms of classification accuracy and CPU runtime (in seconds) from a 10-fold cross validation.

For AODE and NB-Disc, we discretised the attributes using the minimum entropy supervised discretisation method proposed in [3]. BayesNet performs discretisation internally before building the classification model. Runtime includes the discretisation time as well.

We compared the performance of proposed methods with the existing generative classifiers on 18 data sets with different sizes, dimensions, number of classes and class distributions. The properties of the data sets are provided in Table 2.

Table 2: Data sets used to compare the performance of MassCfier with other existing generative classifiers.

Data set	datasize	#dimensions	#classes
CoverType	581012	10	7
MiniBooNE	129596	50	2
OneBig	68000	20	10
Shuttle	58000	8	7
Wave	20000	2	2
RingCurve	20000	2	2
Letters	20000	16	26
Magic04	19020	10	2
Mammography	11183	6	2
Pendigits	10992	16	10
Wine	6497	11	2
Satellite	6435	36	7
OpticalDigits	5620	62	10
PageBlocks	5473	10	5
RobotNavigation	5456	24	4
Waveform	5000	21	3
ImageSegments	2310	19	7
SteelPlateFaults	1941	25	7

Out of 18 data sets used, OneBig, Wave and RingCurve are synthetic and the rest are real data sets. Wave and RingCurve are two dimensional data sets, which are subsets of RingCurve-Wave-TriGaussian data set, shown in Appendix A, each having two classes with 10000 data instances in each class. The OneBig data set [9] has 20 attributes, 9 clusters and 10000 noise instances randomly distributed in the feature space. Noise in the data set are treated as a separate class; hence, it has 10 classes.

5.1 Overall Comparison

5.1.1 Classification Accuracy

The experimental results, in terms of classification accuracies, are show in Table 3.

Compared with existing generative classifiers, the result showed that MassCfier yielded better or at least competitive classification accuracies in most of the data sets. A statistical test based on two standard errors was performed to examine whether the difference is significant. The win:loss:draw counts of MassCfier over existing generative classifiers are reported in Table 4. A win or loss is counted if the difference is significant; otherwise, it is a draw.

Table 3: Classification accuracies (%) on different data sets over a 10-fold cross validation for MassCfier and existing generative classifiers: AODE, BayesNet, NB-KDE, NB-GD and NB-Disc. Figures marked with * and † represent significant win and loss respectively, of MassCfier with respect to AODE based on a two-standard-error significance test.

Data Set	Mass Cfier	AO DE	Bayes Net	NB-KDE	NB-GD	NB-Disc
CoverType	79.16*	72.89	**87.54**	66.72	63.05	66.61
MiniBooNE	**90.90***	89.58	90.19	86.07	83.40	86.29
OneBig	**100***	99.69	99.99	99.98	99.89	99.97
Shuttle	99.89*	99.85	**99.92**	92.68	85.67	94.36
Wave	**99.99***	78.50	78.27	77.91	66.80	78.51
RingCurve	**100***	99.98	99.96	99.27	90.11	99.48
Letters	**96.67***	88.81	86.76	74.21	64.01	73.94
Magic04	**84.58***	83.00	83.36	76.13	72.69	78.30
Mammography	**98.59**	98.42	98.48	97.86	95.68	97.62
Pendigits	**99.45***	97.84	96.56	88.64	85.75	87.90
Wine	**99.32**	99.29	99.20	99.0	97.58	99.01
Satellite	**91.41***	89.26	83.29	82.11	79.52	82.42
OpticalDigits	**98.40***	97.03	96.16	92.21	91.33	92.31
PageBlocks	96.31†	**97.37**	96.25	94.03	90.32	93.57
RobotNavigation	91.51†	94.13	**94.85**	83.39	52.49	88.73
Waveform	84.48†	**86.48**	82.32	80.90	81.04	81.76
ImageSegments	**96.97***	95.76	95.50	85.71	80.17	91.90
SteelPlateFaults	74.19	**75.32**	74.03	62.13	59.25	70.12
Avg. Accuracy	**93.43**	91.29	91.26	85.50	79.93	86.82

MassCfier had 12 wins, 3 losses and 3 draws when compared to AODE, and 11 wins, 2 losses and 5 draws in comparison to BayesNet. Similarly, it had 18 wins over NB-KDE and NB-GD; and 17 wins and 1 draw over NB-Disc.

Table 4: The result of the significance test comparing MassCfier to existing generative classifiers. Values represent win:loss:draw for MassCfier.

	MassCfier
AODE	12:3:3
BayesNet	11:2:5
NB-KDE	18:0:0
NB-GD	18:0:0
NB-Disc	17:0:1

In some data set such as Wave and Letters, MassCfier had significant difference in accuracy than the best existing generative classifier with more than 20% and 8% improvement respectively. In case of Covertype, though the result of MassCfier was significantly poorer than BayesNet, it can be improved by increasing sample size. With $\psi = 16384$, MassCfier yielded accuracy of 87.44% in Covertype.

5.1.2 Run time

Although the runtime results in Table 5 showed that MassCfier does not have the fastest runtime of all classifiers, these results do not present the full picture of the classifiers' time complexities.

Table 5: Run time (in seconds) of a 10-fold cross validation for MassCfier and existing generative classifiers: AODE, BayesNet, NB-KDE, NB-GD, NB-Disc. Runtime for AODE, BayesNet, and NB-Disc includes the time for discretisation as well.

Data Set	Mass Cfier	AO DE	Bayes Net	NB-KDE	NB-GD	NB-Disc
CoverType	1253	105	4208	949	179	96
MiniBooNE	756	162	3230	8160	162	80
OneBig	413	37	4673	2352	30	21
Shuttle	331	8	95	15	10	8
Wave	80	3	2	22	2	2
RingCurve	78	2	4	21	1	1
Letters	425	10	56	23	7	4
Magic04	123	4	14	87	3	4
Mammography	70	2	3	5	2	2
Pendigits	230	5	22	15	4	2
Wine	170	3	5	2	2	2
Satellite	167	8	49	9	4	4
OpticalDigits	175	12	108	9	4	4
PageBlocks	80	2	7	2	4	1
RobotNavigation	143	4	145	22	1	2
Waveform	71	2	7	12	2	1
ImageSegments	89	2	14	4	2	3
SteelPlateFaults	38	2	10	6	1	2

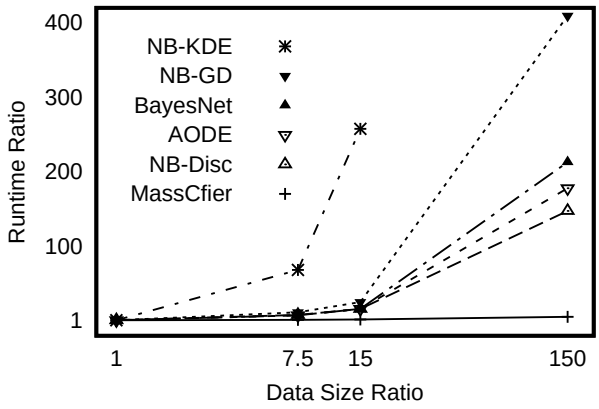
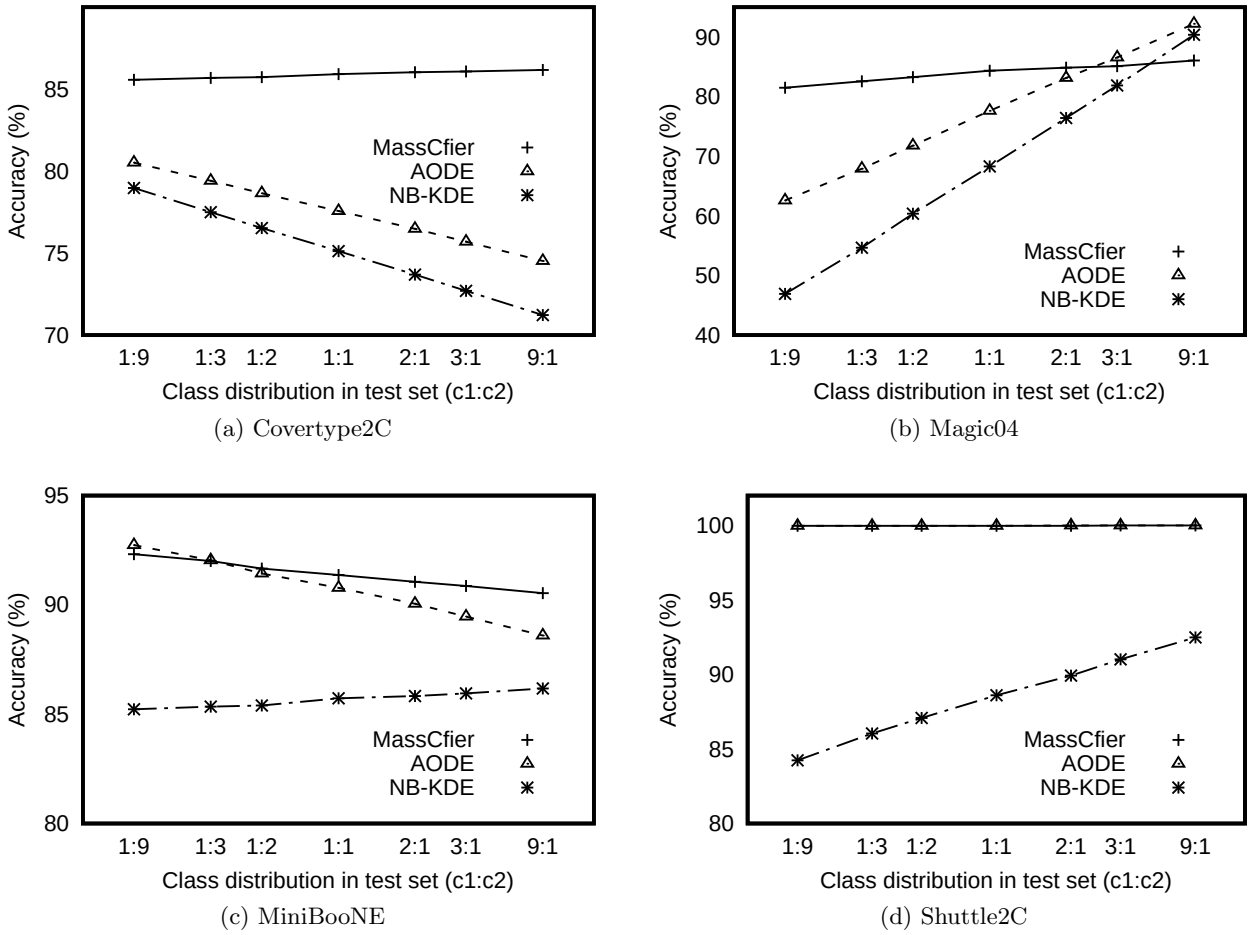


Figure 6: Scale up: MassCfier versus existing generative classifiers in the 48-dimensional Ring-Curve-Wave-TriGaussian data set. The base for data size ratio is 70000 instances and the base for runtime ratio is the runtime on 70000 instances. NB-KDE did not complete in 10 days when data size increased by a factor of 150. Horizontal axis is on logarithmic scale of base 10.

In order to examine how well the classifiers scale-up to large data size, we used the 48-dimensional RingCurve-Wave-TriGaussian data set previously employed by Ting and Wells [16]. It is a combination of three two-dimensional data sets - RingCurve, Wave and TriGaussian as shown in Appendix A, where 42 irrelevant attributes with constant values are added. Data size was increased from 70000 to half-a-million, 1 million and 10 million. Figure 6 showed the increase in runtime of MassCfier and the existing generative classifiers. With the increase in data size by a factor of 7.5 and 15,



(a) Covertype2C (b) Magic04

(c) MiniBooNE (d) Shuttle2C

Figure 7: Influence of class distribution in the test set on classification accuracies of MassCfier, AODE and NB-KDE on Covertype2C, Magic04, MiniBooNE and Shuttle2C data sets. In Shuttle2C, the results of MassCfier and AODE are similar.

MassCfier increased its runtime by a factor of 1.2 and 1.6. The closest contender AODE increased its runtime by a factor of 7 and 16, followed by NB-Disc (8 and 16); BayesNet (8 and 16); NB-GD (11 and 25); and NB-KDE (68 and 258). Even with the data size increase by a factor of 150, MassCfier only increased its runtime by a factor of 6, whereas NB-Disc, AODE, BayesNet and NB-GD increased their runtime by factors of 148, 178, 213 and 409 respectively. NB-KDE did not complete in 10 days when the data size was increased to 10 million. This result showed that MassCfier has a better scale up capabilities than the existing generative classifiers. As MassCfier uses $c \times \psi$ samples only to construct the trees, the increase in the data size n only increases the sampling time. The tree building time is constant.

5.2 Further Analyses

5.2.1 Varying class distribution in the test set

In existing generative classifiers, the prediction is influenced by priors. The accuracy depends upon the proportions of classes in the test data.

In order to examine the effect of varying testing priors on the results of the classifiers, we conducted an experiment on the four largest real data sets - Covertype, MiniBooNE, Shuttle and Magic04. This experiment was conducted using two-class data sets so that the class proportion can be changed easily. Hence, we used a subset of Covertype and Shuttle with the largest two classes - Covertype2C and Shuttle2C. MiniBooNE and Magic04 are two-class data sets. We chose these four largest real data sets so that the range of testing priors can be varied as large as possible.

We did a 10-fold cross validation. In each fold, the actual test set was constructed by random sampling of instances from the original test set of this fold with the desired class distribution. We tested the results of MassCfier and existing generative classifiers: AODE, BayesNet, NB-KDE, NB-GD and NB-Disc. We repeated the experiment with different class distribution in the test set (i.e., c1:c2 = 1:9, 1:3, 1:2, 1:1, 2:1, 3:1, and 9:1).

Figure 7 showed the change in accuracies of MassCfier, AODE and NB-Disc. Table 6 showed the difference between the maximum and minimum accuracies of the classifiers when tested over a range of class distributions in the test set.

From the experiment, it was observed that the accuracies of the existing generative classifiers varied widely across a range of class distribution in the test set. MassCfier produced approximately the same result regardless the propor-

Table 6: Difference between the maximum and minimum accuracies of existing generative classifers and MassCfier over a range of test set with different class distributions (1:9, 1:3, 1:2, 1:1, 2:1, 3:1, 9:1) on four largest real data sets - Covertype, MiniBooNE, Shuttle, Magic04.

Data Set	Mass Cfier	AO DE	Bayes Net	NB-KDE	NB-GD	NB-Disc
Covertype2C	0.60	6.01	4.76	7.75	1.36	7.77
MiniBooNE	1.78	4.12	6.82	0.96	5.2	0.61
Shuttle2C	0.03	0.03	0.03	8.25	31.25	1.50
Magic04	4.54	29.61	18.10	43.44	43.2	38.15

tion of classes in the test data. The results of MassCfier were generally better than that of the existing classifiers. In the case of Magic04, the accuracies of AODE, BayesNet, NaiveBayes varied by 29%, 18% and over 38% respectively; whereas, MassCfier only varies its accuracies by 4.5%. In the case of Covertype2C, AODE, BayesNet, NB-KDE and NB-Disc varied their accuracies by 6%, 4%, 7%, and 7% respectively, but MassCfier produced almost the same results (with variation of 0.6%). In MiniBooNE, MassCfier has slightly higher variation than NB-Disc and NB-KDE. MassCfier, BayesNet and AODE produced the same results in Shuttle2C.

5.2.2 *The effect of discretisation on existing generative classifiers*

The supervised minimum entropy discretisation proposed in [3] was suggested for AODE by the authors of [18]. We tried both supervised minimum entropy discretisation and unsupervised equal frequency bins discretisation [1] in all data sets and observed that AODE, BayesNet and NB-Disc had better results in most cases with supervised discretisation; whereas, in some cases such as Wave, unsupervised technique yielded significantly better results.

To illustrate the effect of the two discretisation techniques, we conducted an experiment on Wave, RingCurve and a subset of Letters and Pendigits data sets with two most misclassified classes - LettersOQ and Digits12. Wave, Letters and Pendigits are the data sets where AODE and BayesNet had significantly poorer results than MassCfier. These data sets are selected to examine whether a different discretisation method would improve the predictive accuracies of AODE, BayesNet and NB-Disc. RingCurve was used to contrast the results.

Table 7 showed the results of AODE, NB-Disc and BayesNet with two different discretisation techniques. In LettersOQ, Digits12 and Wave, both AODE and BayesNet yielded better results with unsupervised techniques; whereas, the supervised technique was better for RingCurve.

As shown in figure 9 (in Appendix A), Wave could not be discretised along horizontal axis with supervised technique. The decision was only based on vertical axis; hence, yielded poor performance. Similarly, in LetterOQ and Digits12, some dimensions could not be discretised with supervised technique and gave poorer performance. With unsupervised techniques, attributes were discretised into 10 equal frequency bins regardless the class distribution that results better density estimation. MassCfier yielded significantly better (based on two-standard error significance test) results than AODE, BayesNet and NB-Disc with either of the discretisation methods.

Table 7: Classification accuracies of AODE, BayesNet and NB-Disc with supervised minimum entropy discretisation [3] and unsupervised equal frequency discretisation with 10 bins [1].

Data sets	Mass Cfier	Supervised Disc. AO DE	Supervised Disc. Bayes Net	Supervised Disc. NB-Disc	Unsupervised Disc. AO DE	Unsupervised Disc. Bayes Net	Unsupervised Disc. NB-Disc
LettersOQ	**98.37**	93.10	95.50	88.15	95.70	96.81	87.37
Digits12	**99.21**	92.87	93.88	87.84	96.24	96.24	87.84
Wave	**99.99**	78.51	78.51	78.51	97.78	97.77	75.80
RingCurve	**100**	99.98	99.98	99.48	98.45	98.45	96.66

5.2.3 *Sensitivity of parameters*

In order to examine the effect of two parameters, sample size (ψ) and number of trees (t) on the classification accuracy of MassCfier, we ran two experiments on four biggest real data sets, namely Covertype, MiniBooNE, Shuttle, and Magic04 varying:

1. Sample size ψ with a fixed number of trees, t =100.
2. Number of trees t with a fixed sample size, ψ =256.

The four largest real data sets were chosen for this experiment so that there are enough instances to vary the sample size (ψ). Figure 8 shows the results of these two experiments.

As sample size ψ was increased, the accuracy increased up to a certain point and then remained flat. But, in case of Covertype, accuracy kept increasing because the two biggest classes have more than two hundred fifty thousand instances each and the continuous improvement in accuracy is a result of improved accuracy for these two classes.

When the number of trees t was increased, the accuracy increased initially and remained constant after reaching a certain point.

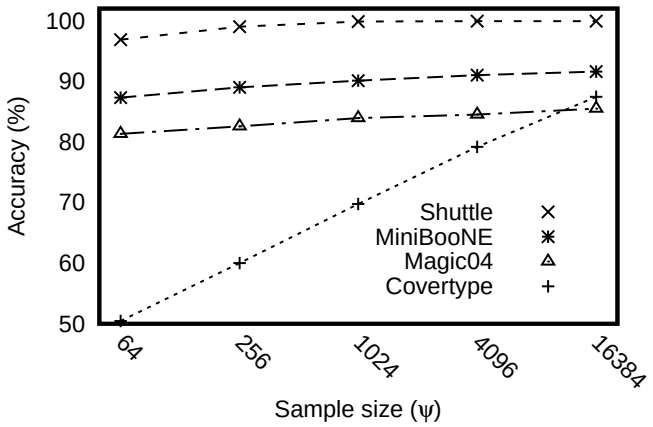
In a nutshell, parameters ψ and t are not very sensitive if set to sufficiently high values.

6. DISCUSSION

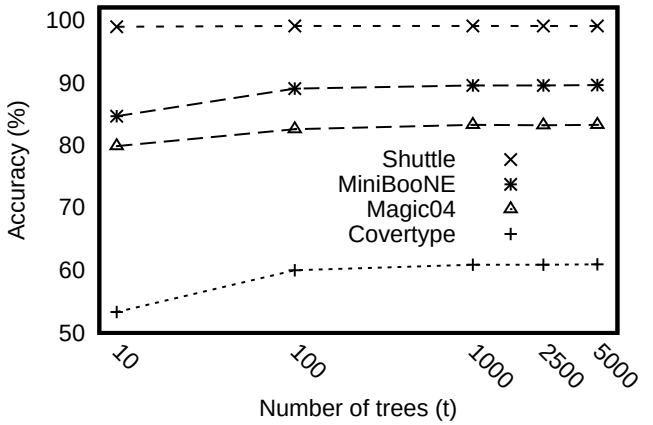
Note that the purpose of trees used in this paper differs from that of decision trees. Trees are used to estimate mass, no class information are used in the building process.

This paper uses the tree-based implementation discussed in [16] for mass estimation. This implementation is grid-based and comes with the weakness of any grid-based methods, especially dealing high dimensional problems. Non-grid based implementations will eliminate the limitations of grid-based implementation and can handle high-dimensional problems more effectively.

Like existing density estimation, mass estimation also requires sufficiently large data set in order to get a good estimation. The bigger the data set the better the mass estimation, and this leads to better classification accuracy. As shown in Table 3 in section 5, MassCfier had 7 wins, 2 draws and 1 loss over the best existing generative classifiers in 10 biggest data sets with data size $n > 10000$ instances.



(a) Effect of sample size (ψ) on the accuracy of MassCfier (t=100). Horizontal axis is on logarithmic scale of base 2. Accuracies were measured on ψ=64, 256, 1024, 4096, and 16384.



(b) Effect of number of trees (t) on the accuracy of MassCfier (ψ=256). Horizontal axis is on logarithmic scale of base 10. Accuracies were measured on t=10, 100, 1000, 2500, and 5000.

Figure 8: Effect of parameters sample size (ψ) and number of trees (t) on the accuracy of MassCfier in four biggest real data sets - CoverType, MiniBooNE, Shuttle and Magic04.

7. CONCLUSION

In this research, we proposed a new type of generative classifier exploiting the notion of mass called **MassCfier**. Unlike existing generative classifiers based on Bayes rule, MassCfier has the following distinctive characteristics.

1. MassCfier estimates the joint distribution directly in multi-dimensional space.
2. MassCfier utilises a new decision rule based on maximum mass rather than Bayes rule.
3. It has sub-linear time complexity and constant space complexity.

Empirical results show that MassCfier is better or at least competitive in terms of classification accuracy when compared to the existing generative classifiers. MassCfier empowers generative classifiers to be more powerful and flexible with no assumption and improved time complexity.
One direction for future work is to explore a non-grid based implementation for mass estimation that eliminates the weaknesses of grid based implementation to deal with high-dimensional problems.

The object code of MassCfier is available at:
https://lore.infotech.monash.edu/
It can be accessed with the following login details:
User name: testuser2
Password: testuser2
The zip file is stored under “File” pane.

8. ACKNOWLEDGEMENT

This project is supported by a grant from the Air Force Research Laboratory, under agreement# FA2386-10-1-4052. Sunil Aryal is supported by a scholarship funded by this grant. The U.S. Government is authorised to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon.

9. REFERENCES

[1] J. Catlett. On changing continuous attributes into ordered discrete attributes. In *Proceedings of the European Working Session on Learning, Porto, Portugal*, pages 164–178, 1991.

[2] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.

[3] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous valued attributes for classification learning. In *Proceedings of 14th International Joint Conference on Artificial Intelligence*, pages 1034–1040, 1995.

[4] A. Frank and A. Asuncion. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2010. University of California, Irvine, School of Information and Computer Sciences.

[5] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29:131–163, 1997.

[6] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. SIGKDD Explorations, 2009. Volume 11, Issue 1.

[7] P. Langley, W. Iba, and K. Thompson. An analysis of bayesian classifiers. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 399–406, 1992.

[8] P. Langley and G. H. John. Estimating continuous distribution in bayesian classifiers. In *Proceedings of Eleventh conference on uncertainty in artificial intelligence*, 1995.

[9] A. Nanopoulos, Y. Theodoridis, and Y. Manolopoulos. Indexed-based density biased sampling for clustering applications. *IEEE Transaction on Data and Knowledge Engineering*, 57(1):37–63, 2006.

[10] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classffiers: A comparison of logistic regression and naive bayes. In *NIPS*, pages 841–848, 2002.

[11] J. Pearl. Bayesian networks: A model of self-activated memory for evedential reasoning. In *Proceedings of the Seventh Annual Conference of the Cognitive Science*

Society, pages 329–334, 1885.

[12] I. Rish. An empirical study of the naive bayes classifier. In *IJCAI Workshop on Empirical Methods in Artificial Intelligence*, 2001.

[13] Y. D. Rubinstein and T. Hastie. Discriminative vs informative learning. In *Proceedings of the Third International Conference on Knowledge and Data Mining*, pages 461–464, 1997.

[14] S. C. Tan, K. M. Ting, and F. T. Liu. Fast anamoly detection for streaming data. In *Proceedings of IJCAI*, pages 1151–1156, 2011.

[15] K. M. Ting, T. Washio, J. R. Wells, and T. Liu. Density estimation based on mass. In *Proceedings of IEEE International Conference on Data Mining*, pages 715–724, 2011.

[16] K. M. Ting and J. R. Wells. Multi-dimensional mass estimation and mass-based clustering. In *Proceedings of IEEE ICDM*, pages 511–520, 2010.

[17] K. M. Ting, G.-T. Zhou, F. T. Liu, and S. C. Tan. Mass estimation and its applications. In *Proceedings of ACM SIGKDD*, pages 989–998, 2010.

[18] G. I. Webb, J. R. Boughton, and Z. Wang. Aggregating one-dependence estimators. *Machine Learning*, 58:5–24, 2005.

[19] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Second Edition. Morgan Kaufmann, 2005.

APPENDIX

A. RINGCURVE-WAVE-TRIGAUSSIAN

The characteristic of Ring-Curve-Wave-TriGaussian data set used in Section 5 for scale-up set is shown in Figure 9. Each of the Ring-Curve, Wave and Triangular-Gaussian, is a two-dimensional data set; and together there is a total of seven classes. Each class has 10000 instances. RingCurve and Wave are also used in performance evaluation in Section 5. When used in the scale up experiment, the data size in each cluster was scaled by a factor of 7.5, 15 to 150.

(a) RingCurve (b) Wave (c) TriGaussian

Figure 9: Scatter plot of RingCurve-Wave-TriGaussian data set, as used in [16], used in Section 5 for scale-up test.

B. ALGORITHMS

We use the same algorithms to generate binary trees to represent $T(\cdot|\mathcal{D})$ as used in [16] for multi-dimensional mass estimation with some minor modifications. Only the pertinent details are provided here. Algorithm 1 generates t trees from a given data set D. Algorithm 2 generates a single tree using a subset $\mathcal{D} \subset D$.

Function $sample(D, \psi)$ samples ψ instances from each class without replacement. If there are not enough instances in a class to sample, then all the instances are picked from that class.

Function InitialiseWorkSpace($\mathcal{D}$), extends the work space that envelops $\mathcal{D}$ by randomly selecting a split point within the range of each dimension and adjusting that split point as the mid-point of the new extended work space.

Algorithm 1 : BuildTrees(D, t, ψ, h)

Inputs: D - input data, t - number of trees, ψ - sub-sampling size, h - number of times an attribute is employed in a path.
Output: F - a set of t h:d-`Trees`

1: $MaxHeightLimit \leftarrow h \times d$
2: **Initialize** F
3: **for** $i = 1$ to t **do**
4: $\quad\mathcal{D} \leftarrow sample(D, \psi)$ {strictly without replacement}
5: $\quad(min,max) \leftarrow$ InitialiseWorkSpace($\mathcal{D}$)
6: $\quad F \leftarrow F \cup$ SingleTree($\mathcal{D}, min, max, 0$)
7: **end for**

Algorithm 2 : SingleTree($\mathcal{D}, min, max, \ell$)

Inputs: $\mathcal{D}$ - input data, min & max - arrays of minimum and maximum values for each attribute in A that define a work space, ℓ - current height level, A - set of d attributes.
Output: an h:d-`Tree`

1: **while** ($\ell < MaxHeightLimit$ and $|\mathcal{D}| > 1$) **do**
2: $\quad$ {Retrieve an attribute from A based on height level.}
3: $\quad q \leftarrow nextAttribute(A, \ell)$
4: $\quad p \leftarrow (max_q + min_q)/2$
5: $\quad \mathcal{D}_l \leftarrow filter(\mathcal{D}, q < p)$
6: $\quad \mathcal{D}_r \leftarrow filter(\mathcal{D}, q \geq p)$
7: $\quad$ **if** ($|\mathcal{D}_l| = 0$) or ($|\mathcal{D}_r| = 0$) **then**
8: $\quad\quad$ {Reduce range for single-branch node.}
9: $\quad\quad$ **if** ($|\mathcal{D}_l| > 0$) **then** $max_q \leftarrow p$
10: $\quad\quad$ **else** $min_q \leftarrow p$
11: $\quad\quad$ **end if**
12: $\quad\quad \ell \leftarrow \ell + 1$
13: $\quad\quad$ continue at the start of while loop
14: $\quad$ **end if**
15: $\quad$ {Build two nodes: $Left$ and $Right$ as a result of a split into two equal-volume half-spaces.}
16: $\quad temp \leftarrow max_q$; $max_q \leftarrow p$
17: $\quad Left \leftarrow$ SingleTree($\mathcal{D}_l, min, max, \ell + 1$)
18: $\quad max_q \leftarrow temp$; $min_q \leftarrow p$
19: $\quad Right \leftarrow$ SingleTree($\mathcal{D}_r, min, max, \ell + 1$)
20: **end while**
21: return $Node(Left, Right, SplitAtt \leftarrow q,$
$\qquad\qquad SplitValue \leftarrow p, Size \leftarrow |\mathcal{D}|)$

Machine Learning manuscript No.
(will be inserted by the editor)

Mass Estimation

Kai Ming Ting · Guang-Tong Zhou · Fei Tony Liu · Tan Swee Chuan

Received: date / Accepted: date

Abstract This paper introduces mass estimation—a base modelling mechanism that can be employed to solve various tasks in machine learning. We present the theoretical basis of mass and efficient methods to estimate mass. We show that mass estimation solves problems effectively in tasks such as information retrieval, regression and anomaly detection. The models, which use mass in these three tasks, perform at least as well as and often better than eight state-of-the-art methods in terms of task-specific performance measures. In addition, mass estimation has constant time and space complexities.

Keywords Mass estimation · density estimation · information retrieval · regression · anomaly detection

1 Introduction

> 'Estimation of densities is a universal problem of statistics (knowing the densities one can solve various problems.)' — V.N. Vapnik [23].

Density estimation has been the base modelling mechanism used in many techniques designed for tasks such as classification, clustering, anomaly detection and information retrieval. For example in classification, density estimation is employed to estimate the class-conditional density function (or likelihood function) $p(x|j)$ or posterior probability $p(j|x)$—the principal function underlying many classification methods; e.g., mixture models, Bayesian networks, Naive Bayes. Examples of density estimation include kernel density estimation, k-nearest neighbours density estimation, maximum likelihood procedures and Bayesian methods.

Rank data points in a given data set in order to differentiate core points from fringe points in a data cloud is fundamental in many tasks, including anomaly detection and information retrieval. Anomaly detection aims to rank anomalous points higher than

K.M. Ting, G.-T. Zhou, F.T. Liu, S.C. Tan
Gippsland School of Information Technology, Monash University, Vic 3842, Australia
E-mail: {kaiming.ting,tony.liu,james.tan}@monash.edu; zhouguangtong@gmail.com

normal points; information retrieval aims to rank points similar to a query higher than dissimilar points. Many existing methods (e.g., [5, 6, 24]) have employed density to provide the ranking; but density estimation is not designed to provide a ranking.

We show in this paper that a new base modelling mechanism called **mass estimation** possesses different properties from those offered by density estimation:

- A mass distribution stipulates an ordering from core points to fringe points in a data cloud. In addition, this ordering accentuates the fringe points with a concave function derived from data, resulting in fringe points have markedly smaller mass than points close to the core points.
- Mass estimation is more efficient than density estimation because mass is computed by simple counting and it requires only a small sample through an ensemble approach. Density estimation (often used to estimate $p(x|j)$ and $p(j|x)$) requires a large sample size in order to have a good estimation and is computationally expensive in terms of time and space complexities [8].

Mass estimation has two advantages in relation to efficacy and efficiency. First, the concavity property mentioned above ensures that fringe points are 'stretched' to be farther from the core points in a mass space—making it easier to separate fringe points from those points close to core points. This property can then be exploited by a machine learning algorithm to achieve a better result for the intended task than the one without it. We show the efficacy of mass in improving the task-specific performance of four existing state-of-the-art algorithms in information retrieval and regression tasks. The significant improvements are achieved through a simple mapping from the original space to a mass space using the mass estimation mechanism introduced in this paper.

Second, mass estimation offers to solve a ranking problem more efficiently using the ordering derived from data directly—without expensive distance (or related) calculation. An example of inefficient application is in anomaly detection tasks where many methods have employed distance or density—a computationally expensive process—to provide the required ranking. An existing state-of-the-art density-based anomaly detector `LOF` [6] (which has quadratic time complexity) completes a job involving half a million data points in more than five hours; yet the mass-based anomaly detector we have introduced here completes it in less than 20 seconds! Section 6.3 provides the details of this example.

The rest of the paper is organised as follows. Section 2 introduces mass and mass estimation, together with their theoretical properties. We also describe methods for one-dimensional mass estimation. We extend one-dimensional mass estimation to multi-dimensional mass estimation in Section 3. We provide the algorithm for multi-dimensional mass estimation in Section 4. Section 5 describes a mass-based formalism which serves as a basis of applying mass to different data mining tasks. We realise the formalism in three different tasks: information retrieval, regression and anomaly detection, and report the empirical evaluation results in Section 6. The relations to kernel density estimation, data depth and other related work are described in Sections 7, 8 and 9, respectively. We provide conclusions and suggest future work in the last section.

2 Mass and mass estimation

Data mass or **mass**, in its simplest form, is defined as the number of points in a region. Any two groups of data in the same domain have the same mass if they have

Table 1 Symbols and notations.

$\mathcal{R}^u$	A real domain of u dimensions
x	A one-dimensional instance in $\mathcal{R}$
$\mathbf{x}$	An instance in $\mathcal{R}^u$
D	A data set of $\mathbf{x}$, where $\|D\| = n$
$\mathcal{D}$	A subset of D, where $\|\mathcal{D}\| = \psi$
$\mathbf{z}$	An instance in $\mathcal{R}^t$
D'	A data set of $\mathbf{z}$
c	The ensemble size used to estimate mass
h	Level of mass distribution
t	Number of mass distributions in $\widetilde{\mathbf{mass}}(\cdot)$
$m_i(\cdot)$	Mass base function defined using binary split s_i
$mass(\cdot)$	Mass function which returns a real value in one-dimensinal mass space
$\widetilde{\mathbf{mass}}(\cdot)$	Mass function which returns a vector of t values in t-dimensional mass space

the same number of points, regardless of the characteristics of the regions they occupy (e.g., density, shape or volume). Mass in a given region is thus defined by a rectangular function which has the same value for the entire region in which the mass is measured.

To estimate the mass for a point and thus the mass distribution of a given data set, a more sophisticated form is required. The intuition is based on the simplest form described above, but multiple (overlapping) regions covering a point are generated. The mass for the point is then derived from an average of masses from all regions covering the point. We show two ways to define these regions. The first is to generate all possible regions through binary splits from the given data points; and the second is to generate random axis-parallel regions, each is within a workspace confined by a data sample. The first is described in this section and the second is described in Section 3.

Each region can be defined in multiple levels where a higher level region covering a point has a smaller volume than that of a lower level region covering the same point. We show that the mass distribution has special properties: (i) the mass distribution defined by level-1 regions is a concave function which has the maximum mass at the centre of the data cloud, irrespective of its density distribution, including uniform and U-shape distributions; and (ii) higher level regions are required to model multi-modal mass distributions.

Note that **mass is not a probability mass function, and it does not provide probability**, as the probability density function does through integration.

In Section 2.1, we show (i) how to estimate a mass distribution for a given data set through binary splits and (ii) the theoretical properties of mass estimation. Section 2.2 describes an approximation to the theoretical mass estimation which works more efficiently in practice. The symbols and notations used are provided in Table 1.

2.1 Mass distribution estimation

In this section, we first show in Section 2.1.1 a mass distribution estimation that uses binary splits in the one-dimensional setting, where each binary split separates the one-dimensional space into two non-empty regions. In Section 2.1.2, we then generalise the treatment using multiple levels of binary splits.

2.1.1 Mass distribution estimation using binary splits

Here, we employ a binary split to divide the data set into two separate regions and compute the mass in each region. The mass distribution at point x is estimated to be the sum of all 'weighted' masses from regions occupied by x, as a result of $n-1$ binary splits for a data set of size n.

Let $x_1 < x_2 < \cdots < x_{n-1} < x_n$ on the real line[1], $x_i \in \mathcal{R}$ and $n > 1$. Let s_i be the binary split between x_i and x_{i+1}, yielding two non-empty regions having two masses m_i^L and m_i^R.

Definition 1 *Mass base function: $m_i(x)$ as a result of s_i, is defined as*

$$m_i(x) = \begin{cases} m_i^L \text{ if } x \text{ is on the left of } s_i \\ m_i^R \text{ if } x \text{ is on the right of } s_i \end{cases}$$

Note that $m_i^L = n - m_i^R = i$.

Definition 2 *Mass distribution: $mass(x_a)$ for a point $x_a \in \{x_1, x_2, \cdots, x_{n-1}, x_n\}$ is defined as a summation of a series of mass base functions $m_i(x)$ weighted by $p(s_i)$ over $n-1$ splits as follows.*

$$\begin{aligned} mass(x_a) &= \sum_{i=1}^{n-1} m_i(x_a) p(s_i) \\ &= \sum_{i=a}^{n-1} m_i^L p(s_i) + \sum_{j=1}^{a-1} m_j^R p(s_j) \\ &= \sum_{i=a}^{n-1} i p(s_i) + \sum_{j=1}^{a-1} (n-j) p(s_j) \end{aligned} \quad (1)$$

$p(s_i)$ is the probability of selecting s_i. Note that we have defined $\sum_{i=q}^{r} f(i) = 0$, when $r < q$ for any function f.

Example. For an example of five points $x_1 < x_2 < x_3 < x_4 < x_5$, Figure 1 shows the resultant $m_i(x)$ due to each of the four binary splits s_1, s_2, s_3, s_4; and their associated masses over four splits are given below:

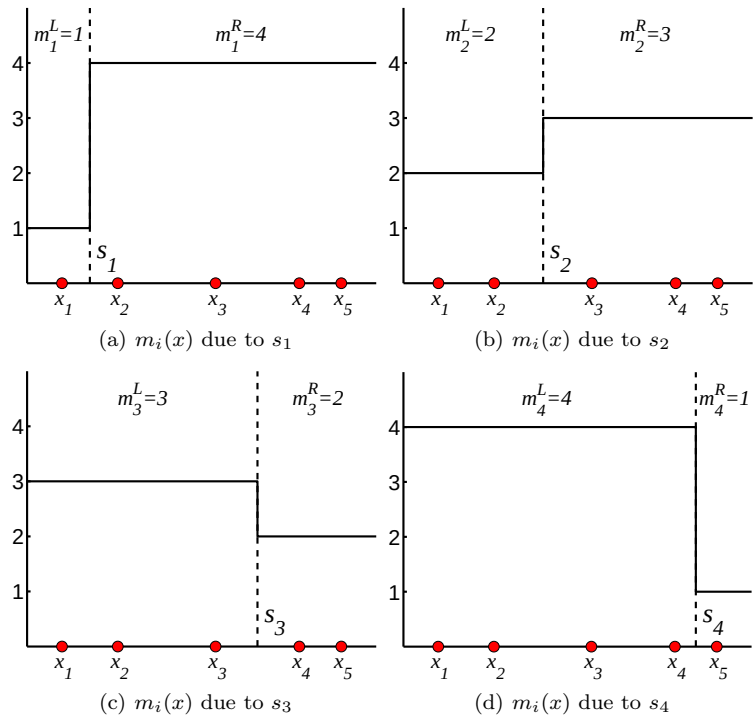
$$\begin{aligned} mass(x_1) &= 1p(s_1) + 2p(s_2) + 3p(s_3) + 4p(s_4) \\ mass(x_2) &= 4p(s_1) + 2p(s_2) + 3p(s_3) + 4p(s_4) \\ mass(x_3) &= 4p(s_1) + 3p(s_2) + 3p(s_3) + 4p(s_4) \\ mass(x_4) &= 4p(s_1) + 3p(s_2) + 2p(s_3) + 4p(s_4) \\ mass(x_5) &= 4p(s_1) + 3p(s_2) + 2p(s_3) + 1p(s_4) \end{aligned}$$

For a given data set, $p(s_i)$ can be estimated on the real line as $p(s_i) = (x_{i+1} - x_i)/(x_n - x_1) > 0$, as a result of random selection of splits based on a uniform distribution.[2]

For a point $x \notin \{x_1, x_2, \cdots, x_{n-1}, x_n\}$, $mass(x)$ is defined as an interpolation between two masses of adjacent points x_i and x_{i+1}, where $x_i < x < x_{i+1}$.

[1] In data having a pocket of points of the same value, an arbitrary order can be 'forced' by adding increasing multiples of an insignificant small value ϵ to each subsequent point of the pocket, without changing the general distribution.

[2] The estimated $mass(x)$ values can be calibrated to a finite data range Δ by multiplying a factor $(x_n - x_1)/\Delta$.



(a) $m_i(x)$ due to s_1 (b) $m_i(x)$ due to s_2

(c) $m_i(x)$ due to s_3 (d) $m_i(x)$ due to s_4

Fig. 1 Examples of mass base function $m_i(x)$ due to each of the four binary splits: s_1, s_2, s_3, s_4.

Theorem 1: $mass(x_a)$ is the maximum at $a = n/2$ for any density distribution of $\{x_1, \cdots, x_n\}$; and the points x_a, where $x_1 < x_2 < \cdots < x_{n-1} < x_n$ on the real line, can be ordered based on mass as follows.

$$\begin{aligned} mass(x_a) &< mass(x_{a+1}),\ a < n/2 \\ mass(x_a) &> mass(x_{a+1}),\ a > n/2 \end{aligned}$$

Proof : The difference in mass between two consecutive points x_a and x_{a+1} differs in only one term, i.e., the mass associated with $p(s_a)$ only; and $\forall i \neq a$, the terms for $p(s_i)$ have the same mass.

$$\begin{aligned} mass(x_a) - mass(x_{a+1}) = & \quad \textstyle\sum_{i=a}^{n-1} ip(s_i) + \sum_{j=1}^{a-1}(n-j)p(s_j) \\ & - \textstyle\sum_{i=(a+1)}^{n-1} ip(s_i) - \sum_{j=1}^{a}(n-j)p(s_j) \\ = & \quad ap(s_a) - (n-a)p(s_a) \\ = & \quad (2a-n)p(s_a) \end{aligned} \tag{2}$$

Thus,

$$sign(mass(x_a) - mass(x_{a+1})) = \begin{cases} negative & \text{if } a < n/2 \\ 0 & \text{if } a = n/2 \\ positive & \text{if } a > n/2 \end{cases}$$

The point $x_{n/2}$ can be regarded as the median. Note that the number of points with the maximum mass depends on whether n is odd or even: When n is an odd integer, only one point has the maximum mass at x_{median}, where $median = \lceil n/2 \rceil$; when n is an even integer, two points have the maximum mass at $a = n/2$ and $a = 1 + n/2$.

□

Theorem 2: $mass(x_a)$ is a concave function defined w.r.t. $\{x_1, x_2, \ldots, x_n\}$, when $p(s_i) = (x_{i+1} - x_i)/(x_n - x_1)$ for $n > 2$.

Proof : We only need to show that the gradient of x_a is non-increasing, i.e., $g(x_a) > g(x_{a+1})$ for each a.

Let $g(x_a)$ be the gradient between x_a and x_{a+1}, and from (2):

$$g(x_a) = \frac{mass(x_{a+1}) - mass(x_a)}{x_{a+1} - x_a} = \frac{n - 2a}{x_n - x_1}$$

The result follows: $g(x_a) > g(x_{a+1})$ for $a \in \{1, 2, \ldots, n-1\}$.

□

Corollary 1 *A mass distribution estimated using binary splits stipulates an ordering, based on mass, of the points in a data cloud from $x_{n/2}$ (with the maximum mass) to the fringe points (with the minimum mass at either side of $x_{n/2}$), irrespective of the density distribution including uniform density distribution.*

Corollary 2 *The concavity of mass distribution stipulates that fringe points have markedly smaller mass than points close to $x_{n/2}$.*

The implication from Corollary 2 is that fringe points are 'stretched' to be farther away from the median in a mass space than in the original space—making it easier to separate fringe points from those points close to the median. (The mass space is mapped from the original space through $mass(x)$.) This property can then be exploited by a data mining algorithm to achieve a better result for the intended task than the one without it. We will show that this simple mapping significantly improves the performance of four existing algorithms in information retrieval and regression tasks in Sections 6.1 and 6.2.

Equation (1) is sufficient to provide a mass distribution corresponding to a unimodal density function or a uniform density function. To better estimate multi-modal mass distributions, a high level of binary splits is required. This is provided in the following.

2.1.2 Level-h mass distribution estimation

Definition 3 *The level-h mass distribution for a point* $x_a \in \{x_1, \ldots, x_n\}$*, where* $h < n$*, is expressed as*

$$\begin{aligned} mass(x_a, h) &= \sum_{i=1}^{n-1} mass_i(x_a, h\text{-}1)p(s_i) \\ &= \sum_{i=a}^{n-1} mass_i^L(x_a, h\text{-}1)p(s_i) + \\ &\quad \sum_{j=1}^{a-1} mass_j^R(x_a, h\text{-}1)p(s_j) \end{aligned} \tag{3}$$

Here a high level mass distribution is computed recursively by using the mass distributions obtained at lower levels. A binary split s_i in a level-$h(>1)$ mass distribution produces two level-(h-1) mass distributions: (a) $mass_i^L(x, h\text{-}1)$—the mass distribution on the left of split s_i which is defined using $\{x_1, \ldots, x_i\}$; and (b) $mass_i^R(x, h\text{-}1)$—the mass distribution on the right which is defined using $\{x_{i+1}, \ldots, x_n\}$. Equation (1) is the mass distribution at level-1.

Figure 2 shows two (out of 19 splits) required to compute level-2 mass estimation, $mass(x, h = 2)$, from a data set of 20 points. Each split produces two level-1 mass estimations: $mass_i^L(x, h = 1)$ and $mass_i^R(x, h = 1)$. Note that level-1 mass distribution is concave, as proven in Theorem 2. This example shows the results of two splits $s_{i=7}$ and $s_{i=11}$, where each level-1 mass distribution is concave.

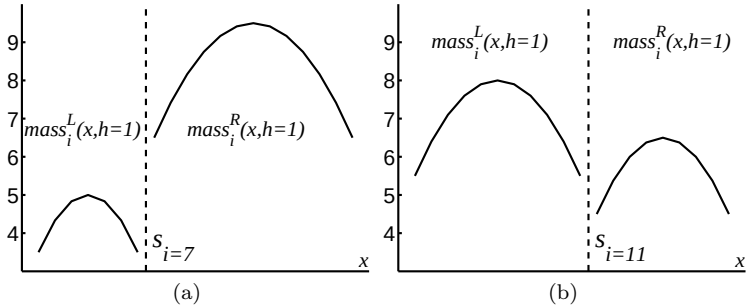


Fig. 2 Two examples of $mass_i^L(x, h = 1)$ and $mass_i^R(x, h = 1)$ due to $s_{i=7}$ and $s_{i=11}$ in the process to get $mass(x, h = 2)$ from a data set of 20 points with uniform density distribution. The resultant $mass(x, h = 2)$ is shown in Figure 3(a).

Using the same analysis as in the proof for Theorem 1, the above equation can be re-expressed as:

$$mass(x_{a+1}, h) = mass(x_a, h) + \begin{cases} [mass_a^R(x_a, h\text{-}1) - mass_a^L(x_a, h\text{-}1)]p(s_a), & h > 1 \\ (n - 2a)p(s_a), & h = 1 \end{cases} \tag{4}$$

As a result, only the mass for the first point x_1 needs to be computed using Equation (3). Note that it is more efficient to compute the mass distribution from the above

equation which has time complexity $O(n^{h+1})$; the computation using Equation (3) has complexity $O(n^{h+2})$.

Definition 4 *A level-h mass distribution stipulates an ordering of the points in a data cloud from α-core points to the fringe points. Each α-core point in a data cloud has the highest mass value within α distance. α-core points are the set of points with $mass^*(h) - mass(x,h) \leq \alpha$, where $mass^*(h) = max_x\ mass(x,h)$. A small α defines local core point(s); and a large α, which covers the entire value range for x, defines global core point(s).*

Examples of level-h mass estimation in comparison with kernel density estimation are provided in Figure 3. Note that $h = 1$ mass estimation looks at the data as a group, and it produces a concave function. As a result, an $h = 1$ mass estimation always has its global core point(s) at the median, regardless of the underlying density distribution—see the four examples of $h = 1$ mass estimation in Figure 3.

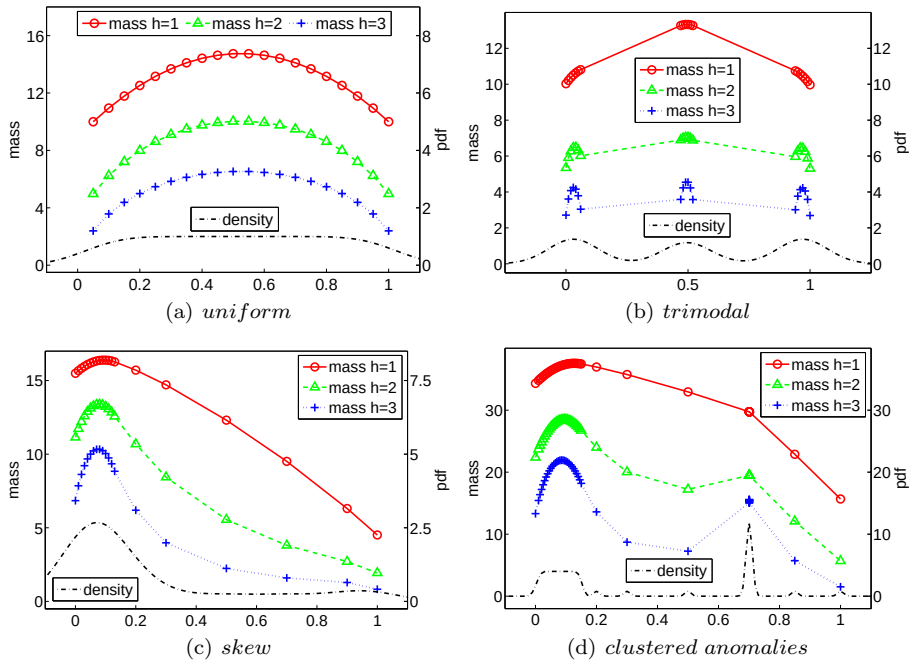


Fig. 3 Examples of level-h mass distribution for $h = 1, 2, 3$ and density distribution from kernel density estimation: Gaussian kernel with bandwidth = 0.1 (for the first three figures) and 0.01 (for the last figure in order to show the density spike.) The data sets have 20 points each for the first three figures, and the last one has 50 points.

For $h > 1$ mass distribution, though there is no guarantee for a concave function any more as a whole, each cluster within the data cloud (if they exist) exhibits a concave function and it becomes more distinct (as a concave function) as h increases. This is shown in Figure 3(b) which has a trimodal density distribution. Notice that the $h > 1$ mass distributions have three α-core points for some α, e.g., 0.2.

Traditionally, one can estimate the core-ness or the fringe-ness of non-uniformly distributed data to some degree by using density or distance (but not in uniform density distribution.) Mass allows one to do that in any distribution without density or distance calculation—the key computational expense in all methods that employ them. For example in Figure 3(c) which has a skew density distribution, the distinction between near fringe points and far fringe points are less obvious using density, unless distances are computed to reveal the difference. In contrast, mass distribution depicts the relative distance from x_{median} using the fringe points' mass values, without further calculation.

Figure 3(d) shows an example where there are clustered anomalies which are denser than the normal points (shown in the bigger cluster on the left of the figure.) Anomaly detection based on density will identify all these clustered anomalies as more 'normal' than the normal points because anomalies are defined as points having low density. In sharp contrast, $h = 1$ mass estimation will correctly rank them as anomalies which have the third lowest mass values. These points are interpreted as points at the fringe of the data cloud of normal points which have higher mass values.

This section has described properties of mass distribution from a theoretical perspective. Though it is possible to estimate mass distribution using Equations (1) and (3), they are limited by its high computational cost. We suggest a practical mass estimation method in the next subsection. We use the term 'mass estimation' and 'mass distribution estimation' interchangeably hereafter.

2.2 Practical one-dimensional level-h mass estimation

Here we devise an approximation to Equation (3) using random subsamples from a given data set.

Definition 5 *$mass(x, h|\mathcal{D})$ is the approximate mass distribution for a point $x \in \mathcal{R}$, defined w.r.t. $\mathcal{D} = \{x_1, \ldots, x_\psi\}$, where $\mathcal{D}$ is a random subset of the given data set D, and $\psi \ll |D|$, $h < \psi$.*

Here we employ a rectangular function to define mass for a region encompassing each point $x \in \mathcal{D}$. $mass(x, h|\mathcal{D})$ is implemented using a lookup table where a region for each point $x_i \in \mathcal{D}$ covers a range $(x_{i-1} + x_i)/2 \leq x < (x_{i+1} + x_i)/2$ and has the same $mass(x_i, h|\mathcal{D})$ value for the entire region. The range for each of the two end-points is set to have equal length on either side of the point. An example is provided in Figure 4(a).

In addition, a number of mass distributions needs to be constructed from different samples in order to have a good approximation, that is,

$$\overline{mass}(x, h) \approx \frac{1}{c} \sum_{k=1}^{c} mass(x, h|\mathcal{D}_k) \tag{5}$$

The computation of $mass(x, h)$ using the given data set D costs $O(|D|^{h+1})$ in terms of time complexity; whereas $mass(x, h|\mathcal{D})$ costs $O(\psi^{h+1})$.

Only relative, not absolute, mass is required to provide an ordering between instances. Because the relative mass is w.r.t. the median and the median is a robust estimator [3]—that is why small subsamples produce a good estimator for ordering.

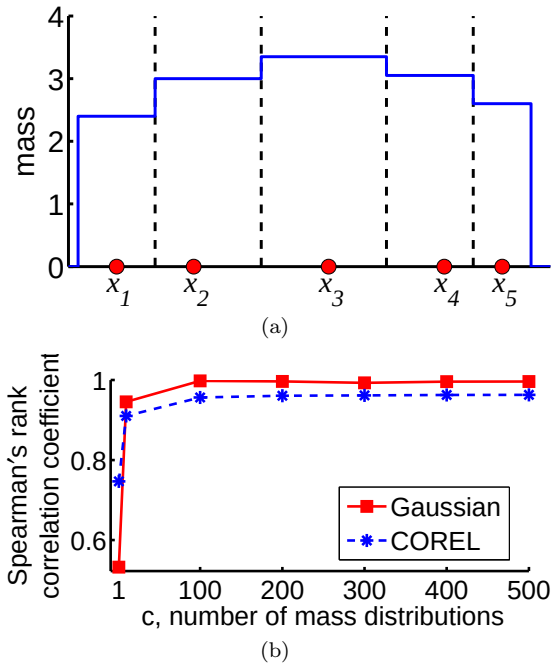


Fig. 4 (a) An example of practical mass distribution $mass(x, h|\mathcal{D})$ for 5 points, assuming a rectangular function for each point. (b) Correlation between the orderings provided by $mass(x, 1)$ and $mass(x, 1|\mathcal{D})$ for two data sets: one-dimensional Gaussian density distribution and the COREL data set used in Section 6.1 (whose result is averaged over 67 dimensions).

Figure 4(b) shows the correlation (in terms of Spearman's rank correlation coefficient) between the orderings provided by $mass(x, 1)$ using the entire data set and $mass(x, 1|\mathcal{D})$ using $\psi = 8$ in two data sets, each having 10000 data points. They achieve very high correlations when $c \geq 100$.

The ability to use a small sample, rather than a large sample, is a key characteristic of mass estimation.

3 Multi-dimensional mass estimation

Here we propose a way to generalise the one-dimensional mass estimation we have described in the last section. It eliminates the need to compute the probability of binary split, $p(s_i)$; and it gives rise to randomised versions of Equations (1), (3) and (5).

The idea is to generate multiple random regions which cover a point, and then the mass for that point is estimated by averaging all masses from all those regions. We show that random regions can be generated using axis-parallel splits called half-space splits. Each half-space split is performed on a randomly selected attribute in a multi-dimensional feature space. For h-level split, each half-space split is carried out h times recursively along every path in a tree structure. Each h-level (axis-parallel) split generates 2^h non-overlapping regions. Multiple h-level splits are used to estimate mass for each point in the feature space.

The multi-dimensional mass estimation requires two functions. First, it needs a function that generates random regions covering each point in the feature space. This function is a generalisation of the binary split into half-space splits or 2^h-region splits when h levels of half-space splits are used. Second, a generalised version of the mass base function is used to define mass in a region. The formal definition follows.

Let $\mathbf{x}$ be an instance in $\mathcal{R}^d$. Let $T^h(\mathbf{x})$ be one of the 2^h regions in which $\mathbf{x}$ falls into; $T^h(\cdot)$ is generated from the given data set D, and $T^h(\cdot|\mathcal{D})$ is generated from $\mathcal{D} \subset D$; and $\mathtt{m}$ be the number of training instances in the region.

The generalised mass base function: $\mathbf{m}(T^h(\mathbf{x}))$ is defined as

$$\mathbf{m}(T^h(\mathbf{x})) = \begin{cases} \mathtt{m} \text{ if } \mathbf{x} \text{ is in a region of } T^h \text{ having } \mathtt{m} \text{ instances,} \\ 0 \text{ otherwise.} \end{cases}$$

In one-dimensional problems, Equations (1), (3) and (5) can now be approximated as follows:

$$\sum_{i=1}^{n-1} m_i(x)p(s_i) \approx \frac{1}{c}\sum_{k=1}^{c} \mathbf{m}(T_k^1(x)) \tag{6}$$

$$mass(x,h) \approx \frac{1}{c}\sum_{k=1}^{c} \mathbf{m}(T_k^h(x)) \tag{7}$$

$$\overline{mass}(x,h) \approx \frac{1}{c}\sum_{k=1}^{c} \mathbf{m}(T_k^h(x|\mathcal{D}_k)) \tag{8}$$

where $c > 0$ is the number of random regions to be used to define mass for x.

Here every T_k^h is generated randomly with equal probability. Note that $p(s_i)$ in Equation (1) has the same assumption.

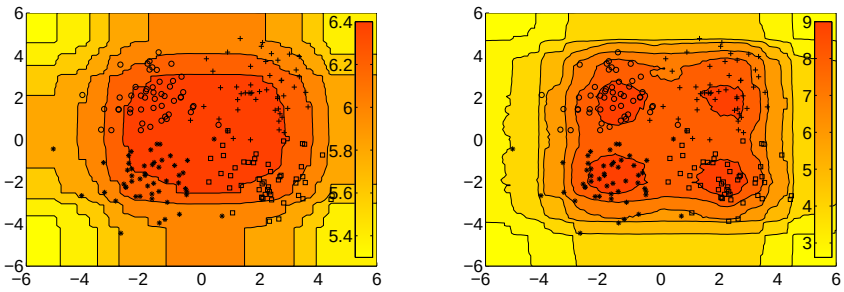
Since T^h is defined in multi-dimensional space, the multi-dimensional mass estimation is the same as Equation (8) by simply replacing x with $\mathbf{x}$:

$$\overline{mass}(\mathbf{x},h) \approx \frac{1}{c}\sum_{k=1}^{c} \mathbf{m}(T_k^h(\mathbf{x}|\mathcal{D}_k)) \tag{9}$$

Like its one-dimensional counterpart, the multi-dimensional mass estimation stipulates an ordering from core points (having high mass) to fringe points (having low mass) in a data cloud, regardless of its density distribution. While we do not have a proof of this property for multi-dimensional mass estimation, empirical results suggest that it is. This property is shown in Figure 5 (a) using $h = 1$, where the highest mass value is at the centre of the entire data cloud, when the four clusters are treated as a single data cloud; while the four clusters are scattered in each of the four quadrants. Mass values become lower as they move away from the centre. Figure 5 (b) shows the contour map for $h = 32$ on the same data set. It demonstrates that multi-dimensional mass estimation can use high h level to model multi-modal distribution.

We show in Section 6 that both $mass(x,h|\mathcal{D})$ and $\mathbf{m}(T^h(\mathbf{x}|\mathcal{D}))$ (in Equations (5) and (9), respectively) can be employed effectively for three different tasks: information retrieval, regression and anomaly detection, through a mass-based formalism to be described in Section 5.

$\mathbf{m}(T^h(\cdot|\mathcal{D}))$ is implemented using Half-Space Trees, first described by Ting, Liu and Tan [19]. There are two variants: The first variant is based on mass only where every external node in a tree has the same depth level. The second variant is based



(a) $h = 1$ (b) $h = 32$

Fig. 5 Contour maps of multi-dimensional mass distribution for a two-dimensional data set with four clusters (each containing 50 points), where points in each cluster are marked with a distinct marker. The points are randomly drawn from Gaussian distributions with unit standard deviation and means located at (2; 2), (-2; 2), (-2; -2) and (2; -2), respectively. The two figures are produced using $h = 1$ and $h = 32$, respectively. Other parameters are set as follows: $c = 1000$ and $\psi = |\mathcal{D}| = 64$. The algorithm used to generate these contour maps will be described in Section 4. The legend indicates the colour-coded mass values.

an augmented mass where the external nodes of a tree have differing depth levels. Because the second variant builds smaller trees and it has similar performance as the first variant, we will use it in this paper. The algorithm for Half-Space Trees and the definition for augmented mass are given in next section. The motivation of Half-Space Trees, described by Ting, Liu and Tan [19], is provided in Appendix A.

4 Algorithm to generate Half-Space Trees

Half-Space Trees estimate a mass distribution efficiently, without density or distance calculations or clustering. We first describe the training procedure, then the testing procedure, and finally the time and space complexities.

Training. The procedure to generate a Half-Space Tree is shown in Algorithm 1. It starts by defining a (random) range for each dimension in order to form a work space which covers all the training data. The **InitialiseWorkSpace**$(\cdot)$ function in Algorithm 1 is carried out as follows. For each attribute q, a random split value (z_q) is chosen within the range $[\mathcal{D}min_q, \mathcal{D}max_q]$, i.e., the minimum and maximum values of q in the subsample. Then, attribute q of the work space is defined to have the range $[min_q, max_q] = [z_q - r, z_q + r]$, where $r = 2 \cdot \max(z_q - \mathcal{D}min_q, \mathcal{D}max_q - z_q)$. The ranges of all dimensions define the work space used to generate a Half-Space Tree. The work space defined by $[min_q, max_q]$ is then passed over to Algorithm 2 to construct a Half-Space Tree.

Constructing a single Half-Space Tree is almost identical to constructing an ordinary decision tree[3] [14], except that no splitting selection criterion is required at each node.

Given a work space, an attribute q is randomly selected to form an internal node of an Half-Space Tree (line 4 in Algorithm 2). The split point of this internal node is

[3] However, they are for different tasks: Decision trees are for supervised learning tasks; Half-Space trees are for unsupervised learning tasks.

simply the mid-point between the minimum and maximum values of attribute q (i.e., min_q and max_q), defined by the work space (line 5). Data are filtered through one of the two branches depending on which side of the split the data reside (lines 6-7). This node building process is repeated for each branch (lines 9-12 in Algorithm 2) until a size limit or a depth limit is reached to form an external node (lines 1-2 in Algorithm 2). The training instances at the external node at depth level ℓ form the mass $\mathbf{m}(T^h(\mathbf{x}|\mathcal{D}))$ to be used during the testing process for $\mathbf{x}$. The parameters are set as follows: $S = log_2(|\mathcal{D}|) - 1$ and $h = |\mathcal{D}|$ for all the experiments conducted in this paper.

Algorithm 1 : $T^h(\mathcal{D}, S, h)$

Inputs: $\mathcal{D}$ - input data, S - data size limit at external node, h - maximum depth limit
Output: a Half-Space Tree

1: $SizeLimit \leftarrow S$
2: $MaxDepthLimit \leftarrow h$
3: (min,max) $\leftarrow$ InitialiseWorkSpace($\mathcal{D}$)
4: **return** SingleHalf-SpaceTree($\mathcal{D}, min, max, 0$)

Algorithm 2 : SingleHalf-SpaceTree($\mathcal{D}, min, max, \ell$)

Inputs: $\mathcal{D}$ - input data, min & max - arrays of minimum and maximum values for all attributes in a work space, ℓ - current depth level
Output: a Half-Space Tree

1: **if** ($|\mathcal{D}| \leq SizeLimit$) or ($\ell \geq MaxDepthLimit$) **then**
2: return $exNode(Size \leftarrow |\mathcal{D}|)$
3: **else**
4: randomly select an attribute q
5: $p \leftarrow (max_q + min_q)/2$
6: $\mathcal{D}_l \leftarrow filter(\mathcal{D}, q < p)$ {Extract data satisfying condition: $q < p$.}
7: $\mathcal{D}_r \leftarrow filter(\mathcal{D}, q \geq p)$ {Extract data satisfying condition: $q \geq p$.}
8: {Build two nodes: $Left$ and $Right$ as a result of a split into two equal-volume half-spaces.}
9: $temp \leftarrow max_q$; $max_q \leftarrow p$
10: $Left \leftarrow$ SingleHalf-SpaceTree($\mathcal{D}_l, min, max, \ell + 1$)
11: $max_q \leftarrow temp$; $min_q \leftarrow p$
12: $Right \leftarrow$ SingleHalf-SpaceTree($\mathcal{D}_r, min, max, \ell + 1$)
13: return $inNode(Left, Right, SplitAtt \leftarrow q,$
$SplitValue \leftarrow p)$
14: **end if**

Ensemble. The proposed method uses a random subsample $\mathcal{D}$ to build one Half-Space Tree (i.e., $T^h(\cdot|\mathcal{D})$), and multiple Half-Space Trees are constructed from different random subsamples (using sampling without replacement) to form an ensemble.

Testing. During testing, a test instance $\mathbf{x}$ traverses through each Half-Space Tree from the root to an external node, and the mass recorded at the external node is used to compute its augmented mass (see Equation (10) below.) This testing is carried out for all Half-Space Trees in the ensemble, and the final score is the average score from all trees, as expressed in Equation (11) below.

The mass, augmented by depth ℓ of the region of Half-Space Tree T^h in which $\mathbf{x}$ falls into, is given as follows.

$$s(\mathbf{x}, h) = \mathbf{m}(T^h(\mathbf{x}|\mathcal{D})) \times 2^\ell \tag{10}$$

Mass needs to be augmented with depth ℓ of a Half-Space Tree in order to 'normalise' the masses from different depths in the tree. See Appendix A for more details.

The mass for point $\mathbf{x}$ estimated from an ensemble of c Half-Space Trees is given as follows.

$$\overline{mass}(\mathbf{x}, h) \approx \frac{1}{c} \sum_{k=1}^{c} s_k(\mathbf{x}, h) \tag{11}$$

Time and Space complexities. Because it involves no evaluations or searches, a Half-Space Tree can be generated quickly. In addition, a good performing Half-Space Tree can be generated using only a small subsample (size ψ) from a given data set of size n, where $\psi \ll n$. An ensemble of Half-Space Trees has training time complexity $O(ch\psi)$ which is constant for an ensemble with fixed subsample size ψ, maximum depth level h and ensemble size c. It has time complexity $O(chn)$ during testing. The space complexity for Half-Space Trees is $O(ch\psi)$ and is also a constant for an ensemble with fixed subsample size, maximum depth level and ensemble size.

5 Mass-based Formalism

The data ordering expressed as mass distribution can be interpreted as a measure of relevance with respect to the concept underlying the data, i.e., points having high mass are highly relevant to the concept and points having low mass are less relevant. In tasks whose primary aim is to rank points in a database with reference to a data profile, mass provides the ideal ranking measure without distance or density calculations. In anomaly detection, high mass signifies normal points and low mass signifies anomalies; in information retrieval, high (low) mass signifies that a database point is highly (less) relevant to the query. Even in tasks whose primary aim is not ranking, the transformed mass space can be better exploited by existing algorithms because the transformation stretches concept-irrelevant points farther away from relevant points in the mass space.

We introduce a formalism in which mass can be applied to different tasks in this section, and provide the empirical evaluation in the following section.

Let $\mathbf{x}_i = [x_i^1, \ldots, x_i^u]$; $\mathbf{x}_i \in D$; and $\mathbf{z}_i = [z_i^1, \ldots, z_i^t]$; $\mathbf{z}_i \in D'$ in the tranformed mass space. The proposed formalism consists of three components:

C1 The first component constructs a number of mass distributions in a mass space. A mass distribution $mass(x^d, h|\mathcal{D})$ for dimension d in the original feature space is obtained using our proposed one-dimensional mass estimation, as given in Definition 5. A total number of t mass distributions is generated which forms $\widetilde{\mathbf{mass}}(\mathbf{x}) \rightarrow \mathcal{R}^t$, where $t \gg u$. This procedure is given in Algorithm 3. Multi-dimensional mass estimation $\mathbf{m}(T^h(\mathbf{x}|\mathcal{D}))$ (replacing one-dimensional mass estimation $mass(x^d, h|\mathcal{D})$) can be used to generate the mass space similarly; see note in Algorithm 3.

C2 The second component maps the data set D in the original space of u dimensions into a new data set D' in t-dimensional mass space using $\widetilde{\mathbf{mass}}(\mathbf{x}) = \mathbf{z}$. This procedure is described in Algorithm 4.

C3 The third component employs a decision rule to determine the final outcome for the task at hand. It is a task-specific decision function applied to $\mathbf{z}$ in the new mass space.

Algorithm 3 : Mass_Estimation(D, ψ, h, t)

Inputs: D - input data; ψ - data size for $\mathcal{D}_k$; h - level of mass distribution; t - number of mass distributions.
Output: $\widetilde{\mathbf{mass}}(\mathbf{x}) \rightarrow \mathcal{R}^t$ - a function consists of t mass distributions, using either one-dimensional or multi-dimensional mass estimation: $mass(x^d, h|\mathcal{D}_k)$ or $\mathbf{m}(T_k^h(\mathbf{x}|\mathcal{D}_k))$.

1: **for** $k = 1$ to t **do**
2: $\quad\mathcal{D}_k \leftarrow$ a random subset of size ψ from D;
3: $\quad d \leftarrow$ a randomly selected dimension from { 1,. . . ,u };
4: $\quad$Build $mass(x^d, h|\mathcal{D}_k)$;
5: **end for**

Note: For multi-dimensional mass estimation, steps 3 and 4 are replaced with one step: Build $\mathbf{m}(T_k^h(\mathbf{x}|\mathcal{D}_k))$;

Algorithm 4 : Mass_Mapping$(D, \widetilde{\mathbf{mass}})$

Inputs: D - input data; $\widetilde{\mathbf{mass}}$ - a function consists of t mass distributions.
Output: D' - a set of mapped instances $\mathbf{z}_i$ in t dimensions.

1: **for** $i = 1$ to $|D|$ **do**
2: $\quad\mathbf{z}_i \leftarrow \widetilde{\mathbf{mass}}(\mathbf{x}_i)$;
3: **end for**

Algorithm 5 : Perform task in `MassSpace`(D, ψ, h, t)

Inputs: D - input data; ψ - data size for $\mathcal{D}$; h - level of mass distribution; t - number of mass distributions.
Output: Task-specific model in mass space.

1: $\widetilde{\mathbf{mass}}(\cdot) \leftarrow$ Mass_Estimation(D, ψ, h, t);
2: $D' \leftarrow$ Mass_Mapping$(D, \widetilde{\mathbf{mass}})$;
3: Perform task (information retrieval or regression) in the mapped mass space using D';

The formalism becomes a blueprint for different tasks. Components **C1** and **C3** are mandatory in the formalism, but component **C2** is optional, depending on the task.

For information retrieval and regression, the task-specific **C3** procedure is simply using an existing algorithm for the task except that the process is carried out in the new mapped mass space, instead of the original space. The `MassSpace` procedure is given in Algorithm 5.

The task-specific **C3** procedure for anomaly detection is shown in steps 2-5 in Algorithm 6: `MassAD`. Note that anomaly detection requires **C1** and **C3** only; whereas the other two tasks require all three components.

In our experiments described in the next section, the mapping from u dimensions to t dimensions using Algorithm 3 is carried out one dimension at a time when using one-dimensional mass estimation; and all u dimensions at a time when using multi-dimensional mass estimation. Each such mapping produces one dimension in mass space

and is repeated t times to get a t-dimensional mass space. Note that randomisation gives different variations to each of the t mappings. The first randomisation occurs at step 2 in Algorithm 3 in selecting a random subset of data. Additional randomisation is applied to attribute selection at step 3 in Algorithm 3 for one-dimensional mass estimation, or at step 4 in Algorithm 2 for multi-dimensional mass estimation.

Algorithm 6 for Anomaly Detection : `MassAD`(D, ψ, h, t)

Inputs: D - input data; ψ - data size for $\mathcal{D}$; h - level of mass distribution; t - number of mass distributions.
Output: Ranked instances in D.
1: $\widetilde{\mathbf{mass}}(\cdot) \leftarrow$ Mass_Estimation(D, ψ, h, t);
2: **for** $i = 1$ to $|D|$ **do**
3: $\quad \mathsf{M}_i \leftarrow$ Average of t masses from $\widetilde{\mathbf{mass}}(\mathbf{x}_i)$;
4: **end for**
5: Rank instances in D based on M_i with low mass denotes anomalies and high mass denotes normal points;

6 Experiments

We evaluate the performance of `MassSpace` and `MassAD` for three tasks in the following three subsections. We denote an algorithm `A` using one-dimensional and multi-dimensional mass estimations as $\mathtt{A}'$ and $\mathtt{A}''$, respectively.

In information retrieval and regression tasks, the mass estimation uses $\psi = 8$ and $t = 1000$. These settings are obtained by examining the rank correlation as shown in Figure 4(b)—having a high rank correlation between $mass(x, 1)$ and $mass(x, 1|\mathcal{D})$. Note that this is done before any method is applied and no further fine-tuning. In anomaly detection tasks, $\psi = 256$ and $t = 100$ are used so that they are comparable to those used in a benchmark method for a fair comparison. In all tasks, $h = 1$ is used for one-dimensional mass estimation, and it cannot afford to use a high h because of its high cost $O(\psi^h)$. $h = \psi$ is used for multi-dimensional mass estimation in order to reduce one parameter setting.

All the experiments were run in Matlab and conducted on a Xeon processor which ran at 2.66 GHz and with 48 GB memory. The performance of each method was measured in terms of task-specific performance measure and runtime. Paired t-tests at 5% significance level were conducted to examine whether the difference in performance is significant between two algorithms under comparison.

Note that we treated information retrieval and anomaly detection as unsupervised learning tasks. Classes/labels in the original data were used as ground truth for evaluation of performance only; they were not used in building mass distributions. In regression, only the training set was used to build mass distributions in step 1 of Algorithm 5; the mapping in step 2 was conducted for both the training and testing sets.

6.1 Content-Based Image Retrieval

We use a Content-Based Image Retrieval (CBIR) task as an example of information retrieval. The `MassSpace` approach is compared with three state-of-the-art CBIR methods that deal with relevance feedbacks: a manifold based method `MRBIR` [11], and two

Table 2 CBIR results (the higher the better for BEP.) An algorithm A using one-dimensional and multi-dimensional mass estimations are denoted as A′ and A″, respectively.

	MRBIR″	MRBIR′	MRBIR	Qsim″	Qsim′	Qsim	InstR″	InstR′	InstR
One query	**12.65**	**10.70**	9.69	**12.38**	**10.35**	7.78	**12.38**	**10.35**	7.78
Round 1	**16.58**	**14.24**	12.72	**19.18**	**15.46**	10.59	**13.88**	**13.33**	9.40
Round 2	**18.41**	**16.05**	13.90	**21.98**	**17.58**	11.81	**15.12**	**14.95**	9.99
Round 3	**19.69**	**17.34**	14.75	**23.67**	**18.71**	12.59	**16.19**	**16.07**	10.36
Round 4	**20.48**	**18.20**	15.33	**24.65**	**19.50**	13.16	**16.88**	**16.93**	10.78
Round 5	**21.15**	**19.86**	15.71	**25.42**	**19.96**	13.55	**17.49**	**17.58**	11.05

recent techniques for improving similarity calculation, i.e., Qsim [27] and InstR [10]; and we employ the Euclidean distance to measure the similarity between instances in these two methods. The default parameter settings are used for all these methods.

Our experiments were conducted using the COREL image database [26] of 10000 images, which contains 100 categories and each category has 100 images. Each image is represented by a 67-dimensional feature vector, which consists of 11 shape, 24 texture and 32 color features. To test the performance, we randomly selected 5 images from each category to serve as the queries. For a query, the images within the same category were regarded as relevant and the rest were irrelevant. For each query, we continued to perform up to 5 rounds of relevance feedback. In each round, 2 positive and 2 negative feedbacks were provided. This relevance feedback process was also repeated 5 times with 5 different series of feedbacks. Finally, the average results with one query and in different feedback rounds were recorded. The retrieval performance was measured in terms of Break-Even-Point (BEP) [27, 26] of the precision-recall curve. The online processing time reported is the time required in each method for a query plus the stated feedback rounds. The reported result is an average over 5×100 runs for query only; and an average over $5 \times 100 \times 5$ runs for query plus feedbacks. The offline costs of constructing the one-dimensional mass estimation and the mapping of 10000 images were 0.27 and 0.32 seconds, respectively. The multi-dimensional mass estimation and the corresponding mapping took 1.72 and 5.74 seconds, respectively.

The results are presented in Table 2 where the retrieval performance better than that conducted in the original space at each round has been boldfaced. The results are grouped for ease of comparison.

The BEP results clearly show that the MassSpace approach achieves a better retrieval performance than that using the original space in all three methods MRBIR, Qsim and InstR, for one query and all rounds of relevance feedbacks. Paired t-tests with 5% significance level also indicate that the MassSpace approach significantly outperforms each of the three methods in all experiments, without exception. These results show that the mass space provides useful additional information that is hidden in the original space.

The results also show that the multi-dimensional mass estimation provides better information than the one-dimensional mass estimation—MRBIR″, Qsim″ and InstR″ give better retrieval performance than MRBIR′, Qsim′ and InstR′, respectively; only some exceptions occur in the higher feedback rounds for InstR′, with minor differences.

The processing time for the MassSpace approach is expected to be longer than each of the three methods because the number of dimensions in the mass space is significantly higher than those in the original space, where $t = 1000$ and $u = 67$. Despite that, MRBIR″, MRBIR′ and MRBIR all have similar level of runtime.

Table 3 CBIR results (time in seconds.)

	MRBIR″	MRBIR′	MRBIR	Qsim″	Qsim′	Qsim	InstR″	InstR′	InstR
One Query	0.714	0.785	0.364	0.715	0.822	0.093	0.715	0.822	0.093
Round1	0.762	0.893	0.696	0.207	0.208	0.035	0.197	0.198	0.026
Round2	0.763	0.893	0.696	0.228	0.231	0.058	0.200	0.200	0.028
Round3	0.763	0.893	0.696	0.257	0.259	0.086	0.200	0.200	0.028
Round4	0.764	0.893	0.696	0.291	0.294	0.122	0.200	0.200	0.028
Round5	0.764	0.893	0.697	0.335	0.341	0.167	0.200	0.200	0.028

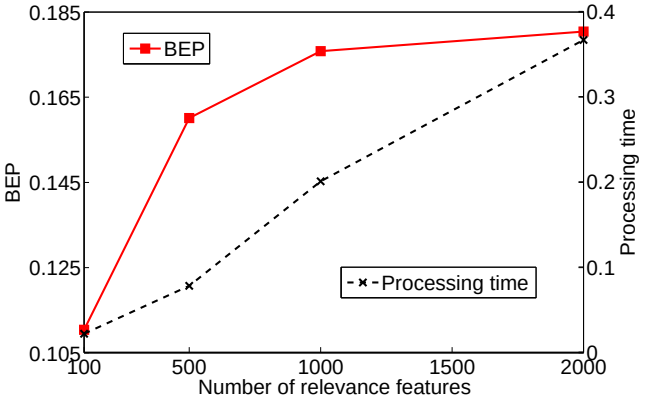


Fig. 6 An example of CBIR-round 5 result: The retrieval performance and the processing time as t increases for InstR′.

Figure 6 shows an example of performance for InstR′—BEP increases as t increases until it reaches a plateau at some t value; and the processing time is linear w.r.t. the number of dimensions of the mass space, t.

6.2 Regression

In this experiment, we compare SVR″ and SVR′ with SVR—support vector regression [23] that employs the mapped mass space versus that employs the original space. SVR is the ϵ-SVR algorithm with RBF kernel, implemented by LIBSVM [7]. SVR is chosen here because it is one of the top performing models.

We utilize five benchmark data sets including four selected from UCI repository [4] and one earthquake data [18] from www.cs.waikato.ac.nz/ml/weka/distribution. The data characteristics are summarized in the first three columns of Table 4. We select only those data sets which are more than 1000 data points with all real-valued attributes and without missing values—in order to get a result with a higher confidence than those obtained from small data sets.

In each data set, we randomly sampled two-thirds of the instances for training and the remaining one-third for testing. This was repeated 20 times and we report the average result of these 20 runs. The data set, whether in the original space or the mass space, was min-max normalized before an ϵ-SVR model was trained. To select optimal parameters for the ϵ-SVR algorithm, we conducted a 5-fold cross validation based on mean squared error using the training set only. The kernel parameter γ was searched in the range $\{2^{-15}, 2^{-13}, 2^{-11}, \cdots, 2^{3}, 2^{5}\}$; the regularization parameter C in the range

Table 4 Regression results (the smaller the better for MSE.)

	data size	MSE ($\times 10^{-2}$)			W/D/L	
		SVR$''$	SVR$'$	SVR	SVR$''$	SVR$'$
tic	9822	**5.56**	**5.58**	5.62	18/0/2	17/0/3
wine_white	4898	**1.08**	**1.21**	1.36	20/0/0	20/0/0
quake	2178	**2.87**	**2.86**	2.92	17/0/3	18/0/2
wine_red	1599	**1.50**	1.62	1.62	19/0/1	11/0/9
concrete	1030	**0.28**	**0.33**	0.57	20/0/0	20/0/0

Table 5 Regression results (time in seconds).

	#Dimension	Processing time			Factor increase		
		SVR$''$	SVR$'$	SVR	time(SVR$''$)	time(SVR$'$)	#dimension
tic	85	23.4	26.6	**11.9**	2.0	2.2	12
wine_white	11	8.2	9.2	**4.2**	2.0	2.2	91
quake	3	2.5	3.4	**1.0**	2.5	3.4	333
wine_red	11	1.7	2.6	**1.0**	1.6	2.5	91
concrete	8	1.2	2.3	**0.9**	1.3	2.6	125

$\{0.1, 1, 10\}$, and ϵ in the range $\{0.01, 0.05, 0.1\}$. We measured regression performance in terms of mean squared error (MSE) and runtime in seconds. The runtime reported is the runtime for SVR only. The total cost of mass estimation (from the training set) and mapping (of training and testing sets) in the largest data set, tic, was 1.8 seconds for one-dimensional mass estimation, and 8.5 seconds for multi-dimensional mass estimation. The cost of normalisation and the parameter search using 5-fold cross-validation was not included in the reported result for all SVR$''$, SVR$'$ and SVR.

The result is presented in Table 4. SVR$'$ performs significantly better than SVR in all data sets in MSE measure; the only exception is in the wine_red data set. SVR$''$ performs significantly better than SVR in all data sets, without exceptions. SVR$''$ generally performs better than SVR$'$.

Although both SVR$''$ and SVR$'$ take more time to run because each of them runs on the data with a significantly higher dimension, yet the factor of increase in time (shown in the last three columns of Table 5) ranges from 1.3 to 3.4 only, when the factor of increase in the number of dimensions ranges from 12 to over 300. This is because the time complexity in the key optimisation process in SVR is not dependent on the number of dimensions.

6.3 Anomaly Detection

This experiment compares MassAD with four state-of-the-art anomaly detectors: isolation forest or iForest [12], a distance-based method ORCA [5], a density-based method LOF [6], and one-class support vector machine (or 1-SVM) [17]. MassAD was built with $t = 100$ and $\psi = 256$, the same default settings as used in iForest [12], which also employed a multi-model approach. The parameter settings employed for ORCA, LOF and 1-SVM were as stated by Liu, Ting and Zhou [12].

All the methods were tested on the eight largest data sets used by Liu, Ting and Zhou [12]. The data characteristics are summarized in Table 6, which include one anomaly data generator Mulcross [15] and the other seven are from UCI repository [4]. The performance was evaluated in terms of averaged AUC (area under ROC curve)

Table 6 Data characteristics of the data sets in anomaly detection tasks. The percentage in brackets indicates the percentage of anomalies.

	data size	#Dimension	anomaly class
Http	567497	3	attack (0.4%)
Forest	286048	10	class 4 (0.9%) vs class 2
Mulcross	262144	4	2 clusters (10%)
Smtp	95156	3	attack (0.03%)
Shuttle	49097	9	classes 2,3,5,6,7 (7%) vs class 1
Mammography	11183	6	class 1 (2%)
Annthyroid	7200	6	classes 1, 2 (7%)
Satellite	6435	36	3 smallest classes (32%)

Table 7 AUC values for anomaly detection.

	MassAD		iForest	ORCA	LOF	1-SVM
	Mass$''$	Mass$'$				
Http	**1.00**	**1.00**	**1.00**	0.36	0.44	0.90
Forest	0.90	**0.92**	0.87	0.83	0.56	0.90
Mulcross	0.26	**0.99**	0.96	0.33	0.59	0.59
Smtp	**0.91**	0.86	0.88	0.87	0.32	0.78
Shuttle	**1.00**	0.99	**1.00**	0.55	0.55	0.79
Mammography	0.86	0.37	**0.87**	0.77	0.71	0.65
Annthyroid	0.75	0.71	**0.82**	0.68	0.72	0.63
Satellite	**0.77**	0.62	0.71	0.65	0.52	0.61

Table 8 Runtime (second) for anomaly detection.

	MassAD		iForest	ORCA	LOF	1-SVM
	Mass$''$	Mass$'$				
Http	168	18	74	9487	18913	35872
Forest	63	10	39	6995	10853	9738
Mulcross	52	10	38	2512	5432	7343
Smtp	27	4	13	267	540	987
Shuttle	20	3	8	157	368	333
Mammography	21	1	3	4	39	11
Annthyroid	7	1	3	2	9	4
Satellite	13	1	3	9	10	9

and processing time (a total of training time and testing time) over ten runs (following [12]).

MassAD and iForest were implemented in Matlab and tested on a Xeon processor ran at 2.66 GHz. LOF was written in Java in ELKI platform version 0.4 [1]; and ORCA was written in C++ (www.stephenbay.net/orca/). The results for ORCA, LOF and 1-SVM were conducted using the same experimental setting but on a slightly slower 2.3 GHz machine, the same machine used by Liu, Ting and Zhou [12].

The AUC values of all the compared methods are presented in Table 7 where the figures boldfaced are the best performance for each data set. The results show that MassAD using the multi-dimensional mass estimation achieves the best performance in four data sets, and close to the best (the difference which is less than 0.03 AUC) in two data sets; MassAD using the one-dimensional mass estimation achieves the best performance in three data sets, and close to the best in one data set. iForest performs best in four data sets. The results are close for these three algorithms because they share many similarities (see Section 9 for details.)

Table 9 Training time and testing time (second) for MassAD and iForest, using $t = 100$ and $\psi = 256$.

	Training time			Testing time		
	MassAD		iForest	MassAD		iForest
	Mass$''$	Mass$'$		Mass$''$	Mass$'$	
Http	16.2	14.3	14.4	151.8	3.3	59.6
Forest	10.3	8.2	8.6	53.1	2.0	30.8
Mulcross	9.1	7.9	8.1	42.8	2.1	29.4
Smtp	5.4	3.9	3.5	21.9	0.6	9.9
Shuttle	6.1	3.1	2.8	14.1	0.3	5.6
Mammography	8.4	1.3	1.2	12.8	0.1	1.8
Annthyroid	3.1	1.3	1.1	3.4	0.1	1.5
Satellite	6.6	1.2	1.6	5.9	0.0	1.9

Again, the multi-dimensional version of MassAD generally performs better than the one-dimensional version, with five wins, one draw and two losses. Most importantly, the worst performance in the Mulcross data set can be easily 'corrected' using a better parameter setting—by using $\psi = 8$, instead of 256, the multi-dimensional version of MassAD improves its detection performance from 0.26 AUC to 1.00 AUC.[4]

It is also noteworthy that the multi-dimensional MassAD significantly outperforms the traditional density-based, distance-based and SVM anomaly detectors in all data sets, except two: one in Annthyroid when compared to ORCA; the poor performance in Mulcross was discussed earlier. The above observations validate the effectiveness of our proposed mass estimation on anomaly detection tasks.

Table 8 shows the runtime result. Although MassAD was run on a slightly faster machine, the result still shows that it has a significant advantage in term of processing time over ORCA, LOF and 1-SVM. The comparison with iForest is presented in Table 9 with a breakdown of training time and testing time. Note that one-dimensional MassAD took the same time as iForest in training, but it only took about one-tenth of the time required by iForest in testing. On the other hand, the multi-dimensional MassAD took slightly more time than iForest in training, but it took up to three times the time required by iForest in testing.

The time and space complexities for five anomaly detection methods are given in Table 10. The one-dimensional MassAD and iForest have the best time and space complexities due to their ability to use small $\psi \ll n$ and $h = 1$. Note that the one-dimensional MassAD ($h = 1$) is faster by a factor of $log(\psi = 256) = 8$ which shows up in the testing time—ten times faster than iForest given in Table 9. The training time disadvantage, compared to iForest, did not show up because of small ψ. The one-dimensional MassAD also has an advantage over iForest in space complexity by a factor of $log(\psi)$. The multi-dimensional MassAD has similar order of worst-case time and space complexities as iForest, though it might have a larger constant.

In contrast to ORCA and LOF (distance-based and density-based methods), the time complexity (and the space complexity) for both MassAD and iForest are independent of the number of dimension u.

[4] Mulcross produces anomaly clusters rather than scattered anomalies. Detecting anomaly clusters are more effective using a low ψ setting when the multi-dimensional version of MassAD is employed.

Table 10 A comparison of time and space complexities. The time complexity includes both training and testing. n is the given data set size and u is the number of dimensions. For `MassAD` and `iForest`, the first part of the summation is the training time and the second the testing time.

	Time complexity	Space complexity
`MassAD` (multi-dimensional)	$O(t(\psi+n)h)$	$O(t\psi h)$
`MassAD` (one-dimensional)	$O(t(\psi^{h+1}+n))$	$O(t\psi)$
`iForest`	$O(t(\psi+n)\cdot log(\psi))$	$O(t\psi\cdot log(\psi))$
`ORCA`	$O(un\cdot log(n))$	$O(un)$
`LOF`	$O(un^2)$	$O(un)$

6.4 Constant time and space complexities

In this section, we show that $mass(x, h|\mathcal{D})$ (in step 4 of Algorithm 3) takes only constant time, regardless of the given data size n, when the algorithmic parameters are fixed. Table 11 reports the runtime time for sampling (to get a random sample of size ψ from the given data set—steps 2 and 3 of Algorithm 3) and the runtime for one-dimensional mass estimation—to construct $mass(x, h|\mathcal{D})$ t times, for five data sets which include the largest and smallest data sets in regression and anomaly detection tasks.

Table 11 Runtime (second) for sampling, $mass(x, 1|\mathcal{D})$ and $mass(x, 3|\mathcal{D})$, where $t = 1000$ and $\psi = 8$.

	data size	sampling	$mass(x, 1\|\mathcal{D})$	$mass(x, 3\|\mathcal{D})$
Http	567497	138.30	0.33	10.96
Shuttle	49097	16.16	0.39	10.97
COREL	10000	1.23	0.27	11.03
tic	9822	1.09	0.43	11.14
concrete	1030	0.18	0.31	10.95

The results show that the sampling time increased linearly with the size of the given data set, and it took significantly longer (in the largest data set) than the time to construct the mass distribution—which was constant, regardless of the given data size. Note that the training time provided in Table 9 includes both the sampling time and mass estimation time, and it is dominated by the sampling time for large data sets.

The memory required for each construction of $mass(x, h|\mathcal{D})$ is to store one lookup table of size ψ which is constant.

The constant time and space complexities apply to multi-dimensional mass estimation too.

6.5 Runtime comparison between one-dimensional and multi-dimensional mass estimations

In terms of runtime, the comparison so far in the experiments might give the impression that multi-dimensional mass estimation is worse than one-dimensional mass estimation.

In fact, the opposite is true because the above results are obtained from $h = 1$ for one-dimensional mass estimation and $h = \psi$ for multi-dimensional mass estimation. Figure 7 shows the head-to-head comparison for different h values in the COREL data set. When h increases from 1 to 5, the runtime for the one-dimensional mass estimation increases by a factor of 33. In contrast, the runtime for the multi-dimensional mass estimation increases by a factor of 1.5 only.

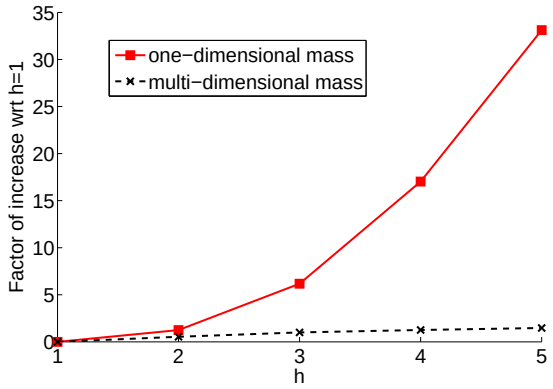


Fig. 7 Runtime comparison: One-dimensional mass estimation versus multi-dimensional mass estimation for different values of h in the COREL data set, where both are using $\psi = 8$ and $t = 1000$. In this experiment, we set h to the required value for multi-dimensional mass estimation, rather than $h = \psi$ which was used in all experiments reported in the previous sections.

Summary

The above results in all three tasks show that the orderings provided by mass distributions deliver additional information about the data that would otherwise hidden in the original features. The additional information, which accentuates fringe points with a concave function, improves the task-specific performance significantly, especially in the information retrieval and regression tasks.

Using Algorithm 5, the runtime is expected to be higher because the new space has much higher dimensions than the original space ($t \gg u$). It shall be noted that the runtime increase (linearly or worse) is solely a characteristic of the existing algorithms used, and is not due to the mass space mapping which has constant time and space complexities.

We believe that a more tailored approach that better integrates the information provided by mass (into the **C3** component in the formalism) for the specific task can potentially further improve the current level of performance in terms of either task-specific performance measure or runtime. We have demonstrated this 'direct' application using Algorithm 6 for the anomaly detection task, in which `MassAD` performs equally well or significantly better than four state-of-the-art methods in terms of task-specific performance measure, and the one-dimensional mass estimation executes faster than all other methods in terms of runtime.

Why does one-dimensional mapping work when tackling multi-dimensional problems? We conjecture that if there is no or little interaction between features, then the one-dimensional mapping will work because the ordering that accentuates the fringe points for each original dimension making it easy for existing algorithms to exploit. When there are strong interactions between features, then one-dimensional mapping might not achieve good results. Indeed, our results in all three tasks show that multi-dimensional mass estimation does perform better than one-dimensional mass estimation in general, in terms of task-specific performance measures.

The ensemble method for mass estimation usually needs only a small sample to build each model in an ensemble. In addition, in order to build all t models for an ensemble, $t\psi$ could be more than n when $\psi > n/t$.

The key limitation of the one-dimensional mass estimation is its high cost when a high value of h is applied. This can be avoided by implementing it using a tree structure rather than a lookup table, as we have done using Half-Space Trees which reduces the time complexity to $O(th(\psi + n))$ from $O(t(\psi^{h+1} + n))$.

7 Relation to Kernel Density Estimation

A comparison of mass estimation and kernel density estimation is provided in Table 12.

Table 12 A comparison of kernel density estimation and mass estimation. Kernel density estimation requires two parameter settings: kernel function $K(\cdot)$ and bandwidth h_w; mass estimation has one: h.

Kernel density$(x) = \frac{1}{nh_w} \sum_{i=1}^{n} K(\frac{x-x_i}{h_w})$
$mass(x,h) = \begin{cases} \sum_{i=1}^{n-1} mass_i(x,h\text{-}1)p(s_i), & h > 1 \\ \sum_{i=1}^{n-1} m_i(x)p(s_i), & h = 1 \end{cases}$

Like kernel estimation, mass estimation at each point is computed through a summation of a series of values from a mass base function $m_i(\cdot)$, equivalent to a kernel function $K(\cdot)$. The two methods differ in the following ways:

- *Aim*: Kernel estimation is aimed to do probability density estimation; whereas mass estimation is to estimate an order from the core points to the fringe points.
- *Kernel function*: While kernel estimation can use different kernel functions for probability density estimation; we doubt that mass estimation requires a different base function for two reasons. First, a more sophisticated function is unlikely to provide a better ordering than a simple rectangular function. Second, the rectangular function keeps the computation simple and fast. In addition, a kernel function must be fixed (i.e., having user-defined values for its parameters); e.g., the rectangular kernel function has fixed width or fixed per unit size. But the rectangular function used in mass has no parameter and no fixed width.
- *Sample size*: Kernel estimation or other density estimation methods require a large sample size in order to estimate the probability accurately [8]. Mass estimation using $mass(x, h|\mathcal{D})$ needs only a small sample size in an ensemble to accurately estimate the ordering.

Here we present the results using a Gaussian kernel density estimation, replacing the one-dimensional mass estimation, using the same subsample size in an ensemble approach. The bandwidth parameter is set to be the standard deviation of the subsample; and all the other parameters are the same.

The results for information retrieval and anomaly detection are provided in Tables 13 and 15. Compared to mass, density performed significantly worse in information retrieval tasks in all experiments using `Qsim` and `InstR`, denoted as $\texttt{Qsim}^K$ and $\texttt{InstR}^K$, respectively. They were even worse than those run in the original space. In anomaly detection, `DensityAD`, which used a Gaussian kernel density estimation, performed significantly worse than `MassAD` in six out of eight data sets in the anomaly detection tasks, and better in the other two data sets.

Table 13 CBIR results (in BEP $\times 10^{-2}$).

(a) Compare with $\texttt{Qsim}^K$ (using kernel density estimation), $\texttt{Qsim}^D$ (using data depth), $\texttt{Qsim}^{LD}$ (using local data depth).

	$\texttt{Qsim}''$	$\texttt{Qsim}'$	$\texttt{Qsim}^K$	$\texttt{Qsim}^D$	$\texttt{Qsim}^{LD}$	Qsim
One Query	**12.38**	10.35	2.90	10.39	7.60	7.78
Round 1	**19.18**	15.46	3.01	15.02	10.95	10.59
Round 2	**21.98**	17.58	2.74	17.16	12.50	11.81
Round 3	**23.67**	18.71	2.54	18.37	13.42	12.59
Round 4	**24.65**	19.50	2.42	19.20	14.03	13.16
Round 5	**25.42**	19.96	2.34	19.74	14.36	13.55

(b) Compare with $\texttt{InstR}^K$, $\texttt{InstR}^D$ and $\texttt{InstR}^{LD}$.

	$\texttt{InstR}''$	$\texttt{InstR}'$	$\texttt{InstR}^K$	$\texttt{InstR}^D$	$\texttt{InstR}^{LD}$	InstR
One Query	**12.38**	10.35	2.90	10.39	7.60	7.78
Round 1	**13.88**	13.33	2.91	13.05	8.71	9.40
Round 2	**15.12**	14.95	2.55	14.73	9.68	9.99
Round 3	**16.19**	16.07	2.25	15.98	10.28	10.36
Round 4	16.88	**16.93**	2.06	16.82	10.78	10.78
Round 5	17.49	**17.58**	1.99	17.50	11.17	11.05

Table 14 CBIR results (time in seconds).

(a) Compare with $\texttt{Qsim}^K$, $\texttt{Qsim}^D$, $\texttt{Qsim}^{LD}$.

	$\texttt{Qsim}''$	$\texttt{Qsim}'$	$\texttt{Qsim}^K$	$\texttt{Qsim}^D$	$\texttt{Qsim}^{LD}$	Qsim
One Query	0.715	0.822	0.820	0.840	0.829	0.093
Round 1	0.207	0.208	0.224	0.237	0.226	0.035
Round 2	0.228	0.231	0.279	0.288	0.276	0.058
Round 3	0.257	0.259	0.348	0.355	0.343	0.086
Round 4	0.291	0.294	0.435	0.438	0.425	0.122
Round 5	0.335	0.341	0.547	0.543	0.531	0.167

(b) Compare with $\texttt{InstR}^K$, $\texttt{InstR}^D$ and $\texttt{InstR}^{LD}$.

	$\texttt{InstR}''$	$\texttt{InstR}'$	$\texttt{InstR}^K$	$\texttt{InstR}^D$	$\texttt{InstR}^{LD}$	InstR
One Query	0.715	0.822	0.820	0.840	0.829	0.093
Round 1	0.197	0.198	0.203	0.215	0.206	0.026
Round 2	0.200	0.200	0.205	0.216	0.206	0.028
Round 3	0.200	0.200	0.206	0.217	0.207	0.028
Round 4	0.200	0.200	0.207	0.218	0.208	0.028
Round 5	0.200	0.200	0.207	0.218	0.208	0.028

Table 15 Anomaly detection: `MassAD` vs `DensityAD` and `DepthAD` (AUC).

	MassAD		DensityAD	DepthAD	
	$\texttt{Mass}''$	$\texttt{Mass}'$		Depth	LDepth
Http	**1.00**	**1.00**	0.99	0.98	0.52
Forest	0.90	**0.92**	0.70	0.85	0.49
Mulcross	0.26	0.99	**1.00**	0.99	0.93
Smtp	0.91	0.86	0.59	0.92	**0.93**
Shuttle	**1.00**	0.99	0.90	0.87	0.72
Mammography	**0.86**	0.37	0.27	0.36	0.79
Annthyroid	0.75	0.71	0.80	0.58	**0.86**
Satellite	**0.77**	0.62	0.61	0.59	0.69

Table 16 Anomaly detection: `MassAD` vs `DensityAD` and `DepthAD` (time in seconds).

	MassAD		DensityAD	DepthAD	
	$\texttt{Mass}''$	$\texttt{Mass}'$		Depth	LDepth
Http	168	18	17	38	38
Forest	63	10	10	31	31
Mulcross	52	10	10	31	31
Smtp	27	10	10	26	26
Shuttle	20	4	4	25	25
Mammography	21	3	3	24	24
Annthyroid	7	1	1	23	23
Satellite	13	1	1	23	23

8 Relation to Data Depth

There is a close relationship between the proposed mass and data depth [13]: they both delineate the centrality of a data cloud (as opposed to compactness in the case of the density measure.) The properties common to both measures are: (a) the centre of a data cloud has the maximum value of the measure; (b) an ordering from the centre (having the maximum value) to the fringe points (having the minimum values).

However, there are two key differences. First, not until recently (see [2]) data depth always models a given data with one centre, regardless whether the data is unimodal or multi-modal; whereas mass can model both unimodal and multi-modal data by setting $h = 1$ or $h > 1$. Local data depth [2] has a parameter (τ) which allows it to model multi-modal data as well as unimdal data. However, the performance of local data depth appears to be sensitive to the setting of τ (see a discussion of the comparison below.) In contrast, a single setting of h in mass estimation had produced good task-secific performance in three different tasks in our experiment.

Second, mass is a simple and straightforward measure, and has efficient estimation methods based on axis-parallel partitions only. Data depth has many different definitions, depending on the construct used to define depth. The constructs could be Mahalanobis, Convex Hull, simplicial, halfspace and so on [13], all of which are expensive to compute [3]—this has been the main obstacle in applying data depth for real applications in multi-dimensional problems. For example, Ruts and Rousseeuw [16] compute the contour of data depth of a data cloud for visualization, and employ depth as the anomaly score to identify anomalies. Because of its computational cost, it is limited to small data size only. In contrast to the axis-parallel partitions used in mass

estimation, halfspace data depth[5] [22], for example, requires to consider all halfspaces which demands high computational time and space.

To provide a comparison, we replace the one-dimensional mass estimation (defined in Algorithm 3) with data depth (defined by simplicial depth [13]). We repeat the experiments by employing the data depth implementation in R by Agostinelli and Romanazzi [2] (accessible from r-forge.r-project.org/projects/localdepth.) Data depth is carried out in the same approach by using sample size ψ to build each of the t models in an ensemble [6]. The number of simplices used to do the empirical estimation is set to 10000 for all runs. Default settings are used for all other parameters (i.e., the membership of a data point in simplices is evaluated in the "exact" mode rather than the approximate mode, and the tolerance parameter is fixed to 10^{-9}). Note that local depth uses an additional parameter τ to select candidate simplices, where a simplex volumed larger than τ is excluded from consideration. As the performance of local depth is sensitive to τ, we employ the quantile order of τ of 10%, the low value of the range 10%-30% suggested by Agostinelli and Romanazzi [2]. Because both data depth and local data depth are estimated using the same procedure, their runtimes are the same.

The task-specific performance results for information retrieval and anomaly detection are provided in Tables 13 and 15. Note that local data depth could produce worse retrieval results than those in the original feature space. Data depth performs close to that achieved by the one-dimensional mass estimation, but is significantly worse than the multi-dimensional mass estimation. In anomaly detection, data depth performs worse than both versions of mass estimation in six out of eight data sets; local data depth performs worse than multi-dimensional mass estimation in five out of eight data sets; local data depth versus one-dimensional mass estimation have four wins and four losses. Note that though local data depth achieves the best result in two data sets, it also produces the worst in three data sets which are significantly worse than others (in http, forest and shuttle.)

The runtime results are provided in Tables 14 and 15. These results do not reveal the time complexities of the algorithms. We conducted a scale up test using the Mulcross data set by increasing the subsampling size ψ from 8 to 4096, doubling at each step increase. The result is presented in Figure 8. It shows that data depth or local data depth had the worst runtime ratio which increased its runtime 58 times when ψ was increased by a factor of 512. The multi-dimensional mass estimation had the best runtime ratio of 6.6, followed by KDE (24) and one-dimensional mass estimation (34)

[5] Zuo and Serfling [28] define halfspace data depth (HD) of a point x in $\mathcal{R}^u$ w.r.t. a probability measure P on $\mathcal{R}^u$ as the minimum probability mass carried by any closed halfspace containing x:

$$HD(x;P) = inf\{P(H) : H \text{ a closed halfspace}, x \in H\}, x \in \mathcal{R}^u$$

In the language of data depth, the one-dimensional mass estimation may be interpreted as a kind of average probability mass of halfspaces containing x, weighted by mass covered by halfspace. But the one-dimensional mass estimation defined in Equation (1) allows mass to be computed by a summation of $n-1$ components from the given data set of size n, whereas data depth does not. In addition, our implementation of multi-dimensional mass estimation using a tree structure with axis-parallel splits cannot be interpreted using any of the constructs employed by data depth.

[6] Our experiments indicate that using the entire data set to estimate data depth or local data depth produces worse results than those using an ensemble approach. This result is shown in Appendix B.

when ψ ratio = 512. The actual runtimes in seconds are 126.6 (Mass$''$), 166.7 (KDE), 239.4 (Mass$'$), and 600.5 (Data Depth). This result shows that the multi-dimensional mass estimation has the best time complexity and data depth has the worst time complexity among the four algorithms.

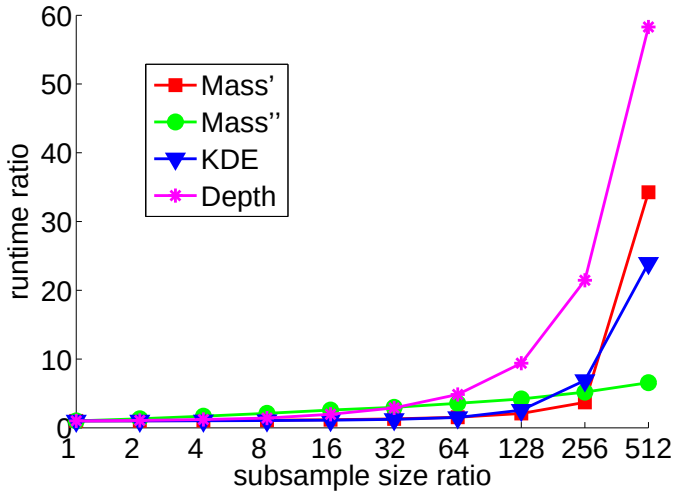


Fig. 8 Scale up test using the Mulcross data set. The base subsampling data size (ψ) is 8; doubling at each step until $\psi = 4096$. Each point in the graph is an average over 10 runs.

9 Other work based on mass

`iForest` [12] and `MassAD` share some common features: Both are ensemble methods which build t models, each from a random sample of size ψ, and they both combine the outputs of the models through averaging during testing. Although `iForest` [12] employs path length—an instance traverses from the root of a tree to its leaf—as the anomaly score, we have shown that the path length used in `iForest` is in fact a proxy to mass [19] (see Appendix A for a brief description.) In other words, `iForest` is a kind of mass-based method—that is why `MassAD` and `iForest` have similar detection accuracy. Multi-dimensional `MassAD` has the closest resemblance to `iForest` because of the use of tree. The key difference is that `MassAD` is just one application of the more fundamental concept of mass introduced here, whereas `iForest` is for anomaly detection only. The trees in `iForest` are completely random whereas Half-Space Trees are partially random.

How easily can the proposed formalism be applied to other tasks? In addition to the tasks we have applied in this paper, we have applied mass estimation 'directly', using the proposed formalism, to solve problems in content-based multimedia information retrieval [25] and clustering [20]. While the 'indirect' application is straightforward which simply uses the existing algorithms in the mass space, a 'direct' application requires a complete rethink of the problem and produces a totally different algorithm.

However, this rethink of a problem in terms of mass often results a more efficient and sometimes more effective algorithm than existing algorithms. We provide a brief description of the two applications in the following two paragraphs.

In addition to the mass-space mapping we have shown here (i.e., components **C1** and **C2**), Zhou et al [25] present a content-based information retrieval method that assigns a weight (based on `iForest`, thus, mass) to each new mapped feature w.r.t. a query; and then it ranks objects in the database according to their weighted average feature values in the mapped space. The method also incorporates relevance feedback which modifies the ranking based on the feedbacks through reweighted features in the mapped space. This method forms the third component of the formalism stated in Section 5. This 'direct' application of mass has been shown to be significantly better than the 'indirect' approach we have shown in Section 6.1, in terms of both task-specific measure and runtime [25]. It is interesting to note that, unlike existing retrieval systems which rely on a metric, the new mass-based method does not employ a metric—it is the first information retrieval system that does not use a metric, as far as we know.

Ting and Wells [20] use a variant of Half-Space Trees we have employed here and apply mass directly to solve clustering problems. It is the first mass-based clustering algorithm, and it is unique because it does not use any distance and density measure. In this task, like in the case of anomaly detection, only two components are required. After building a mass model (in the **C1** component), the **C3** component consists of linking instances with non-zero mass connected by the mass model and making each group of connected instances a separate cluster; and all other unconnected instances are regarded as noise. This mass-based clustering algorithm has been shown to perform equally well as DBSCAN [9] in terms of clustering performance, but it runs orders of magnitude faster [20].

The earlier version of this paper [21] establishes the properties of mass estimation in the one-dimensional setting only; and use it in all three tasks. This paper extends one-dimensional mass estimation to multi-dimensional mass estimation using the same approach as described by Ting and Wells [20], and implements multi-dimensional mass estimation using Half-Space Trees [19]. This paper reports new experiments using the multi-dimensional mass estimation, and shows the advantage of using multi-dimensional mass estimation over one-dimensional mass estimation in the three tasks reported earlier [21]. These related works show that mass estimation can be implemented in different ways using tree-based or non-tree-based methods.

10 Conclusions and future work

This paper makes two key contributions. First, we introduce a base measure, mass, and delineate its three properties: (i) a mass distribution stipulates an ordering from core points to fringe points in a data cloud; (ii) this ordering accentuates the fringe points with a concave function—a property that can be easily exploited by existing algorithms to improve their task-specific performance; and (iii) the mass estimation methods have constant time and space complexities. Density estimation has been the base modelling mechanism employed in many techniques thus far. Mass estimation introduced here provides an alternative choice, and it is better suited for many tasks which require an ordering rather than probability density estimation.

Second, we present a mass-based formalism which forms a basis to apply mass for different tasks. The three tasks (i.e., information retrieval, regression and anomaly

detection) in which we have successfully applied are just examples of its application. Mass estimation has potentials in many other applications.

There are potential extensions to the current work. First, the algorithms for the three tasks and the formalism can be improved or extended to include more tasks. Second, because the purposes and their properties differ, mass estimation is not intended to replace density estimation—it is thus important to identify areas in which each is best suited for. This will ascertain (i) areas in which density has been a mismatch, unbeknown up to now, and (ii) areas in which mass estimation is weak. We postulate that density does not perform as well as mass in our experiments is because of a lack of concavity. This will be determined in the future. Third, the current implementation of multi-dimensional mass estimation using Half-Space Trees limits its applications to low dimensional problems because it suffers the same problem as in all other grid oriented methods. We will explore non-grid oriented implementations of mass which have potential to tackle high dimensional problems more effectively than existing density-based and distance-based methods.

The Matlab source code of mass estimation is available at http://sourceforge.net/projects/mass-estimation/.

References

1. E. Achtert, H.-P. Kriegel, and A. Zimek ELKI: A Software System for Evaluation of Subspace Clustering Algorithms In *Proceedings of the 20th International Conference on Scientific and Statistical Database Management*, pages 580-585, 2008.
2. C. Agostinelli and M. Romanazzi. Local depth. *Journal of Statistical Planning and Inference* 141, pages 817-830. 2011.
3. G. Aloupis. Geometric measures of data depth. *DIMACS Series in Discrete Math and Theoretical Computer Science*, 72:147–158, 2006.
4. A. Asuncion and D. Newman. UCI machine learning repository, 2007.
5. S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of ACM SIGKDD*, pages 29–38, 2003.
6. M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying density-based local outliers. In *Proceedings of ACM SIGKDD*, pages 93–104, 2000.
7. C.-C. Chang and C.-J. Lin. LIBSVM: a library for support vector machines, 2001.
8. R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Second Edition. John Wiley, 2001.
9. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of ACM SIGKDD*, pages 226–231, 1996.
10. G. Giacinto and F. Roli. Instance-based relevance feedback for image retrieval. In *Advances in NIPS*, pages 489–496, 2005.
11. J. He, M. Li, H. Zhang, H. Tong, and C. Zhang. Manifold-ranking based image retrieval. In *Proceedings of ACM Multimedia*, pages 9–16, 2004.
12. F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *Proceedings of IEEE ICDM*, pages 413–422, 2008.

13. R. Liu, J. M. Parelius, and K. Singh. Multivariate analysis by data depth. *The Annals of Statistics*, 27(3):783–840, 1999.
14. J. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.
15. D. M. Rocke and D. L. Woodruff. Identification of outliers in multivariate data. *Journal of the American Statistical Association*, 91(435):1047–1061, 1996.
16. I. Ruts and P. Rousseeuw. Computing depth contours of bivariate point clouds. *Computational Statistics and Data Analysis*, 23(1):153–168, 1996.
17. B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt. Support vector method for novelty detection. In *Advances in NIPS*, pages 582–588, 2000.
18. J. S. Simonoff. *Smoothing Methods in Statistics*. Springer-Verlag, 1996.
19. K. M. Ting, S. C. Tan, and F. T. Liu. Mass: A new ranking measure for anomaly detection. *Gippsland School of Information Technology, Monash University*, Technical Report TR2009/1, 2009.
20. K. M. Ting and J. R. Wells. Multi-dimensional mass estimation and mass-based clustering. In *Proceedings of IEEE ICDM*, pages 511–520, 2010.
21. K. M. Ting, G.-T. Zhou, F. T. Liu, and S. C. Tan. Mass estimation and its applications. In *Proceedings of ACM SIGKDD*, pages 989–998, 2010.
22. J.W. Tukey. Mathematics and picturing data. In *Proceedings of the International Congress on Mathematics*. 2, pages 525-531, 1975.
23. V. N. Vapnik. *The Nature of Statistical Learning Theory*. Second Edition. Springer, 2000.
24. R. Zhang, and Z. Zhang. BALAS: Empirical Bayesian Learning in the Relevance Feedback for Image Retrieval. *Image and Vision Computing*, 24(3): 211-223, 2006.
25. G.-T. Zhou, K. M. Ting, F. T. Liu, and Y. Yin. Relevance feature mapping for content-based multimedia information retrieval. *Pattern Recognition*, 45: 1707-1720, 2012.
26. Z.-H. Zhou, K.-J. Chen, and H.-B. Dai. Enhancing relevance feedback in image retrieval using unlabeled data. *ACM Transactions on Information Systems*, 24(2):219–244, 2006.
27. Z.-H. Zhou and H.-B. Dai. Query-sensitive similarity measure for content-based image retrieval. In *Proceedings of IEEE ICDM*, pages 1211–1215, 2006.
28. Y. Zuo and R. Serfling. General notion of statistical depth function. *The Annal of Statistics* 28:461-482, 2000.

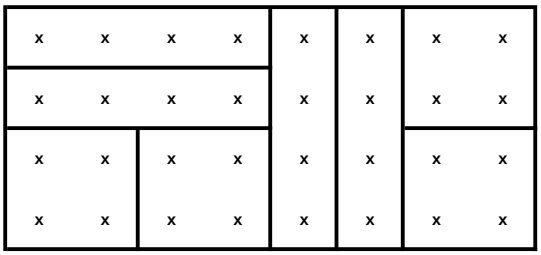
Appendix A: Half-Space Trees for Mass Estimation

This section describes the implementation of T^h using Half-Space Tree. Two variants are provided. We have used the second variant of Half-Space Tree to implement the multi-dimensional mass estimation.

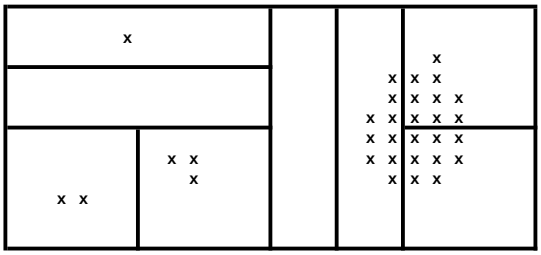
Half-Space Tree

The motivation of the proposed method, Half-Space Tree, comes from the fact that equal-size regions contain the same mass in a space with uniform mass distribution, regardless of the shapes of the regions. This is shown in Figure 9(a), where the space

enveloped by the data is split into equal-size half-spaces recursively three times into eight regions. Note that the shapes of the regions may be different because the splits at the same level may not use the same attribute.



(a) Uniform mass distribution.



(b) Non-uniform mass distribution.

Fig. 9 Half-space subdivisions of: (a) uniform mass distribution; and (b) non-uniform mass distribution.

The binary half-space split ensures that every split produces two equal-size half-spaces, each containing exactly half of the mass before the split under a uniform mass distribution. This characteristic enables us to compute the relationship between any two regions easily. For example, the mass in every region shown in Figure 9(a) is the same, and it is equivalent to the original mass divided by 2^3 because three levels of binary half-space splits have been applied. A deviation from the uniform mass distribution allows us to rank the regions based on mass. Figure 9(b) provides such an example in which a ranking of regions based on mass provides an order of the degrees of anomaly in each region.

Definition 6 *: Half-Space Tree is a binary tree in which each internal node makes a half-space split into two equal-size regions, and each external node terminates further splits. All nodes record the mass of the training data in their own regions.*

Let $T^h[i]$ be a Half-Space Tree with depth level i; and $\mathbf{m}(T^h[i])$ or short for $\mathbf{m}[i]$ be the mass in one of the regions at level i.

The relationship between any two regions is expressed using mass with reference to $\mathbf{m}[0]$ at depth level=0 (the root) of a Half-Space Tree.

Under uniform mass distribution, the mass at level i is related to mass at level 0 as follows:

$$\mathbf{m}[0] = \mathbf{m}[i] \times 2^i,$$

or mass values between any two regions at levels i and j, $\forall i \neq j$, are related as follows:

$$\mathbf{m}[i] \times 2^i = \mathbf{m}[j] \times 2^j.$$

Under non-uniform mass distribution, the following inequality establishes an ordering between any two regions at different levels:

$$\mathbf{m}[i] \times 2^i < \mathbf{m}[j] \times 2^j.$$

We employ the above property to rank instances and define the (augmented) mass for Half-Space Tree as follows.

$$s(\mathbf{x}) = \mathbf{m}[\ell] \times 2^\ell, \tag{12}$$

where ℓ is the depth level of an external node with $\mathbf{m}[\ell]$ instances in which a test instance $\mathbf{x}$ falls into.

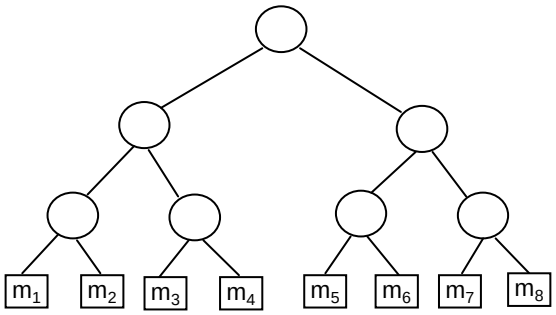
Mass is estimated using $\mathbf{m}[\ell]$ only if a Half-Space Tree has all external nodes at the same depth level. The estimation is based on augmented mass, $\mathbf{m}[\ell] \times 2^\ell$, if the external nodes have differing depth levels. We describe two such variants of Half-Space Tree below.

HS-Tree: based on mass only. The first variant, HS-Tree, builds a balanced binary tree structure which makes a half-space split at each internal node and all external nodes have the same depth. The number of training instances falling into each external node is recorded and it is used for mass estimation. An example of HS-Tree is shown in Figure 10(a).

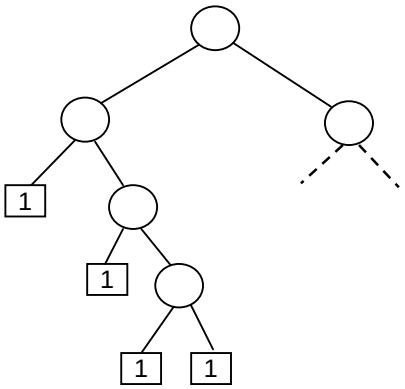
HS*-Tree: based on augmented mass. Unlike HS-Tree, the second variant, HS*-Tree, whose external nodes have differing depth levels. The mass estimated from HS*-Tree is Equation (1) in order to account for different depths. We call this *augmented mass* because the mass is augmented in the calculation by the depth level in HS*-Tree, as opposed to mass only in HS-Tree.

In a special case of HS*-Tree, the tree growing process at a branch will terminate to form an external node when the training data size at the branch is 1 (i.e., the size limit is set to 1.) Here the mass estimated depends on depth level only, i.e., 2^ℓ or simply ℓ. In other words, **the depth level becomes a proxy for mass in HS*-Tree** when the size limit is set to 1. An example of HS*-Tree, when the size limit is set to 1, is shown in Figure 10(b).

Since the two variants have similar performance, we focus on HS*-Tree only in this paper because it builds a smaller size tree than HS-Tree which may grow many branches with zero mass—this saves on training time and memory space requirements.



(a) HS-Tree.



(b) HS*-Tree.

Fig. 10 Half-Space Tree: (a) HS-Tree: An HS-Tree for the data shown in Figure 9a has $\mathtt{m}_i = 4, \forall i$, which are $\mathbf{m}[\ell = 3]$ (i.e., mass at level 3). (b) HS*-Tree: An example of a special case of HS*-Tree when the size limit is set to 1.

Appendix B: Anomaly detection using data depth that builds a single model from the entire data set

This appendix provides the results in anomaly detection task where data depth and local data depth built a single model from the entire data set, i.e., $\texttt{DepthAD}_s$. This is in contrast to `DepthAD` which employed an ensemble approach in Section 8.

Table 17 shows that `MassAD` generally has higher AUC than $\texttt{DepthAD}_s$ which employed either data depth or local data depth. The only exception is the Annthyroid data set. Note that these resutls are generally worse than those employing an ensemble approach, reported in Table 15.

Knowledge and Information Systems

Density Estimation based on Mass

--Manuscript Draft--

Manuscript Number:	
Full Title:	Density Estimation based on Mass
Article Type:	SC: IEEE ICDM 2011 (by Invitation)
Keywords:	density estimation; density-based algorithms
Corresponding Author:	Sunil Aryal Monash University, Gippsland Campus Churchill, Victoria AUSTRALIA
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Monash University, Gippsland Campus
Corresponding Author's Secondary Institution:	
First Author:	Kai Ming Ting, Ph.D
First Author Secondary Information:	
All Authors:	Kai Ming Ting, Ph.D
	Takashi Washio, Ph.D
	Jonathan R. Wells
	Fei Tony Liu, Ph.D
	Sunil Aryal
All Authors Secondary Information:	
Abstract:	Density estimation is the ubiquitous base modelling mechanism employed for many tasks including clustering, classification, anomaly detection and information retrieval. Commonly used density estimation methods such as kernel density estimator and k-nearest neighbour density estimator have high time and space complexities which render them inapplicable in problems with large data size and even a moderate number of dimensions. This weakness sets the fundamental limit in existing algorithms for all these tasks. We propose the first density estimation method which stretches this fundamental limit to an extent that dealing with millions of data can now be done easily and quickly. We analyse the error of the new estimation (from the true density) using a bias-variance analysis. We then perform an empirical evaluation of the proposed method by replacing existing density estimators with the new one in three current density-based algorithms, namely, DBSCAN, LOF and Bayesian classifier, representing three different data mining tasks of clustering, anomaly detection and classification respectively. The results show that the new density estimation method significantly improves their time complexities, while maintaining or improving their task-specific performances in clustering, anomaly detection, and classification respectively. The new method empowers these algorithms, currently limited to small data size only, to process very large databases --- setting a new benchmark for what density-based algorithms can achieve.

Density Estimation based on Mass

Kai Ming Ting*, Takashi Washio†, Jonathan R. Wells*, Fei Tony Liu* and Sunil Aryal*
*Gippsland School of Information Technology, Monash University,
Gippsland Campus, Churchill, Victoria 3842, Australia
Email: {kaiming.ting, jonathan.wells, tony.liu, sunil.aryal}@monash.edu
†The Institute of Scientific and Industrial Research, Osaka University,
8-1 Mihogaoka, Ibarakishi, Osaka, 5670047, Japan.
Email: washio@ar.sanken.osaka-u.ac.jp

***Abstract*—Density estimation is the ubiquitous base modelling mechanism employed for many tasks including clustering, classification, anomaly detection and information retrieval. Commonly used density estimation methods such as kernel density estimator and k-nearest neighbour density estimator have high time and space complexities which render them inapplicable in problems with large data size and even a moderate number of dimensions. This weakness sets the fundamental limit in existing algorithms for all these tasks.**

We propose the first density estimation method which stretches this fundamental limit to an extent that dealing with millions of data can now be done easily and quickly. We analyse the error of the new estimation (from the true density) using a bias-variance analysis. We then perform an empirical evaluation of the proposed method by replacing existing density estimators with the new one in three current density-based algorithms, namely, DBSCAN, LOF and Bayesian classifier, representing three different data mining tasks of clustering, anomaly detection and classification respectively. The results show that the new density estimation method significantly improves their time complexities, while maintaining or improving their task-specific performances in clustering, anomaly detection, and classification respectively. The new method empowers these algorithms, currently limited to small data size only, to process very large databases — setting a new benchmark for what density-based algorithms can achieve.

***Keywords*-density estimation; density-based algorithms;**

I. Introduction

Density estimation is ubiquitously applied to various tasks such as clustering, classification, anomaly detection and information retrieval. Despite its pervasive use ('estimation of densities is a universal problem of statistics' [22]), there are no efficient density estimation methods thus far. Most existing methods such as kernel density estimator and k-nearest neighbour (k-NN) density estimator cannot be applied to problems with even a moderate number of dimensions and large data size. This paper is motivated to introduce the first efficient method for density estimation. We show that three existing density-based algorithms, when employ the new density estimator, set a new runtime benchmark that is orders of magnitude faster. For example, two of these algorithms now take only days instead of months to complete tasks involving millions of instances, after the existing density estimators are replaced with the new one.

We make four contributions in this paper:

1) Propose a new density estimation method which has a significant advantage over existing methods in terms of time and space complexities.
2) Establish the characteristics of the method through a bias-variance analysis.
3) Verify the generality of the method by replacing existing density estimators with the new one in three current density-based algorithms.
4) Significantly simplify and speed up the current algorithms using set-based definitions instead of the common point-based definitions (see Section V.)

The new density estimation method distinguishes itself from existing methods by:

- Employing no distance measures in the density estimation process.
- Having average case sublinear time complexity and constant space complexity. Thus, it can be applied to very large databases in which current methods such as kernel and k-NN density estimators are infeasible because they are prohibitively expensive to compute.

Two existing density estimators are presented in Section II, in order to contrast with the new density estimator we introduce in Section III. We analyse the error produced by the new estimator by a bias-variance analysis and provide a comparison of the estimation results between the new estimator and kernel density estimator in Section IV. Sections V and VI describe how the new estimator can replace existing density estimators in three current state-of-the-art density-based algorithms and their empirical evaluation results, respectively. A discussion of the related issues and the conclusions are provided in the last two sections.

II. Density Estimation

This section describes two, probably the most commonly used, density estimation methods, namely kernel density estimator and k-nearest neighbour density estimator.

A. Kernel Density Estimator

Let $\mathbf{x}$ be an instance in a d-dimensional space $\mathcal{R}^d$. The kernel density estimator (KDE) defined by a kernel function

$K(\cdot)$ and bandwidth b is given as follows [17].

$$\bar{f}_{KDE}(\mathbf{x}) = \frac{1}{nb^d} \sum_{i=1}^{n} K(\frac{\mathbf{x} - \mathbf{x}_i}{b})$$

The difference $\mathbf{x} - \mathbf{x}_i$ requires some form of distance measure; and n is the number of instances in the given data set D. An example of $K(\cdot)$, as a rectangular function, is given as follows.

$$K(\mathbf{x}) = \begin{cases} \frac{1}{2} & \text{if } |\mathbf{x}| < 1 \\ 0 & \text{otherwise.} \end{cases}$$

B. k-NN Density Estimator

A k-nearest neighbour (k-NN) density estimator can be expressed as follows [18].

$$\bar{f}_{kNN}(\mathbf{x}) = \frac{|N(\mathbf{x}, k)|}{n \sum_{\mathbf{x}' \in N(\mathbf{x},k)} distance(\mathbf{x}, \mathbf{x}')}$$

where $N(\mathbf{x}, k)$ is the set of k nearest neighbours to $\mathbf{x}$; and the search for nearest neighbours is conducted over D of size n.

Both KDE and k-NN density estimators have $O(n^2)$ time complexity and $O(n)$ space complexity in order to estimate the densities of n instances. Although there are various indexing schemes to speed up the search for nearest neighbour in order to aid the k-NN density estimator, they are not satisfactory in terms of dealing with high dimensional problems and large data sets. We will provide further discussion of this issue in Section VII.

III. Density estimator based on mass

A recently introduced base measure called mass [21] has demonstrated its wide application to solve various data mining tasks such as regression, information retrieval, clustering and anomaly detection, including one in data stream [21], [20], [19].

Because mass is more fundamental than density, we show in this paper that a density estimator can be constructed from mass. The key advantage of mass is that it can be computed very quickly. The new density estimator based on mass inherits this advantage and executes significantly faster than existing density estimators such as KDE and k-NN. It raises the capability of density-based algorithms to handle large data sets to a new high level.

A mass base function is defined as follows by [20]

$$\mathbf{m}(T(\mathbf{x})) = \begin{cases} \mathsf{m} & \text{if } \mathbf{x} \text{ is in a region of } T(\cdot), \\ 0 & \text{otherwise,} \end{cases}$$

where $T(\cdot)$ is function which subdivides the feature space into non-overlapping regions based on the given data set D; and m is the number of samples in a region of $T(\mathbf{x})$ in which $\mathbf{x}$ falls into.

Ting and Wells [20] shows that mass can also be effectively estimated using data subsets $\mathcal{D}_i \subset D$ $(i = 1, \ldots, t)$ and its associated $T_i(\mathbf{x}|\mathcal{D}_i)$, where $|\mathcal{D}_i| = \psi \ll n$. Each $\mathcal{D}_i$ is sampled without replacement from D. The mass estimated using subsamples is defined as

$$\overline{mass}(\mathbf{x}) = \frac{1}{t} \sum_{i=1}^{t} \mathbf{m}(T_i(\mathbf{x}|\mathcal{D}_i)).$$

We now introduce the new density estimators based on mass (DEMass) and describe its implementation in the next two subsections.

A. DEMass

Once mass is estimated, density can be estimated as a ratio of mass and volume.

Thus, the new density estimators based on mass functions $\mathbf{m}(T(\mathbf{x}))$ and $\mathbf{m}(T_i(\mathbf{x}|\mathcal{D}_i))$ are defined respectively as

$$f_{\mathbf{m}}(\mathbf{x}) = \frac{\mathbf{m}(T(\mathbf{x}))}{nv}. \tag{1}$$

$$\bar{f}_{\mathbf{m}}(\mathbf{x}) = \frac{1}{t} \sum_{i=1}^{t} \frac{\mathbf{m}(T_i(\mathbf{x}|\mathcal{D}_i))}{\psi v_i}. \tag{2}$$

where v and v_i are the volumes of regions $T(\mathbf{x})$ and $T_i(\mathbf{x}|\mathcal{D}_i)$, respectively.

We use the term DEMass to refer to density estimator $\bar{f}_{\mathbf{m}}(\mathbf{x})$ in the rest of this paper.

DEMass has two key differences/advantages when compared to the one based on a kernel method or k-NN:

- $\bar{f}_{\mathbf{m}}$ is estimated from $t\psi$ instances only which are significantly smaller than D in a large data set. It sums over t number of randomly generated regions; whereas $\bar{f}_{KDE}$ sums over n number of instances in D, and $\bar{f}_{kNN}$ also requires a search on the entire data set. For a large data set, $\bar{f}$ is prohibitively expensive to compute in these two methods [1].
- $\bar{f}_{\mathbf{m}}$ needs no distance measures.

B. Implementation

Mass estimation can be implemented in different ways [21], [20], [19].

When $T(\cdot|\mathcal{D})$ is implemented using a binary tree, the volumes of regions in $T(\cdot|\mathcal{D})$ are controlled by a parameter h which defines the level of binary subdivision.

Let Δ_i be a work space in $\mathcal{R}^d$ which envelops $\mathcal{D}_i$; and Δ_i has its length along each dimension j as $\Delta_{ij} = \max(x_{kj}|\mathbf{x}_k \in \mathcal{D}_i) - \min(x_{kj}|\mathbf{x}_k \in \mathcal{D}_i)$. Each $T_i(\cdot|\mathcal{D}_i)$ is constructed within work space Δ_i, resulting in 2^{hd} hyper-rectangular regions where every region has an equi-width $\delta x_{ij} = \Delta_{ij}/2^h$ on each dimension j and a volume $v_i =$

[1] While there are ways to reduce the computational cost of KDE and k-NN, they are usually limited to low dimensional problems or incur significant preprocessing cost. See Section VII for a discussion.

$\delta x_{i1} \times \cdots \times \delta x_{id}$. For example, in a one-dimensional space with work space Δ_i derived from $\mathcal{D}_i$ and $h = 3$, $T_i(\cdot|\mathcal{D}_i)$ subdivides the work space into 2^3 equi-width regions. We use T_i to denote T_i^h, unless h is required in the context; and T_i is built from $\mathcal{D}_i$, for each i.

We use the implementation of $T(\cdot|\mathcal{D})$ as described in [20] as the basis to build density estimator $\bar{f}_{\mathbf{m}}$.

$T(\mathbf{x}|\mathcal{D})$ is represented as a binary tree (called h:d-Tree in [20]), where each path from the root to a leaf has $h \times d$ nodes such that each of the d attributes appears exactly h times.

Let $\ell = h \times d$, and m_k be the mass of region k. There is a total of 2^ℓ regions which have a total mass: $|\mathcal{D}| = \sum_{k=1}^{2^\ell} \mathsf{m}_k$, where $\mathsf{m}_k = \mathbf{m}(T(\mathbf{x}|\mathcal{D}))$; and $\mathbf{x}$ is in region k of T.

Algorithm 1 generates t trees from a given data set D. Algorithm 2 generates a single tree using a subset $\mathcal{D} \subset D$, where $|\mathcal{D}| = \psi$.

Algorithm 1 : BuildTrees(D, t, ψ, h)

Inputs: D - input data, t - number of trees, ψ - subsampling size, h - number of times an attribute is employed in a path.
Output: F - a set of t h:d-Trees

1: $MaxHeightLimit \leftarrow h \times d$
2: **Initialise** F
3: **for** $i = 1$ to t **do**
4: $\quad \mathcal{D} \leftarrow sample(D, \psi)$ {strictly without replacement}
5: $\quad (min, max) \leftarrow$ InitialiseWorkSpace($\mathcal{D}$)
6: $\quad F \leftarrow F \cup$ SingleTree$(\mathcal{D}, min, max, 0)$
7: **end for**

The time complexity of constructing the trees is $O(t\psi hd)$. The space complexity is $O(thd + n)$ during construction. After the trees are built, the data set is discarded, yielding $O(thd)$.

To estimate the density of a given instance $\mathbf{x}$, only these trees are used according to Equation (2).

In the next section, we will show that the bias between $\bar{f}_{\mathbf{m}}(\mathbf{x})$ and the true probability density function $p_d(\mathbf{x})$ converges asymptotically.

IV. Error Analysis through Bias-Variance Decomposition

The density estimator based on mass (DEMass) $\bar{f}_{\mathbf{m}}(\mathbf{x})$ can be thought of as a random variable because of its dependence on D and its random subsamples $\mathcal{D}_i$ $(i = 1, \ldots, t)$. Accordingly, we analyse Mean Squared Error (MSE) of $\bar{f}_{\mathbf{m}}(\mathbf{x})$ from its true probability density $p_d(\mathbf{x})$. It is defined as

$$MSE(\bar{f}_{\mathbf{m}}(\mathbf{x})) = E\big[\{\bar{f}_{\mathbf{m}}(\mathbf{x}) - p_d(\mathbf{x})\}^2\big]$$

where the expectation $E[\cdot]$ is taken over the distribution of $\bar{f}_{\mathbf{m}}(\mathbf{x})$. This is rewritten by introducing the expectation of

Algorithm 2 : SingleTree$(\mathcal{D}, min, max, \ell)$

Inputs: $\mathcal{D}$ - input data, min & max - arrays of minimum and maximum values for each attribute in A that define a work space, ℓ - current height level, A - set of d attributes.
Output: an h:d-Tree

1: **while** $(\ell < MaxHeightLimit)$ **do**
2: $\quad$ {Retrieve an attribute from A based on height level.}
3: $\quad q \leftarrow nextAttribute(A, \ell)$
4: $\quad p \leftarrow (max_q + min_q)/2$
5: $\quad \mathcal{D}_l \leftarrow filter(\mathcal{D}, q < p)$
6: $\quad \mathcal{D}_r \leftarrow filter(\mathcal{D}, q \geq p)$
7: $\quad$ **if** $(|\mathcal{D}_l| = 0)$ or $(|\mathcal{D}_r| = 0)$ **then**
8: $\quad\quad$ {Reduce range for single-branch node.}
9: $\quad\quad$ **if** $(|\mathcal{D}_l| > 0)$ **then** $max_q \leftarrow p$
10: $\quad\quad$ **else** $min_q \leftarrow p$
11: $\quad\quad$ **end if**
12: $\quad\quad \ell \leftarrow \ell + 1$
13: $\quad\quad$ continue at the start of while loop
14: $\quad$ **end if**
15: $\quad$ {Build two nodes: $Left$ and $Right$ as a result of a split into two equal-volume half-spaces.}
16: $\quad temp \leftarrow max_q$; $max_q \leftarrow p$
17: $\quad Left \leftarrow$ SingleTree$(\mathcal{D}_l, min, max, \ell + 1)$
18: $\quad max_q \leftarrow temp$; $min_q \leftarrow p$
19: $\quad Right \leftarrow$ SingleTree$(\mathcal{D}_r, min, max, \ell + 1)$
20: **end while**
21: return *Node*$(Left, Right, SplitAtt \leftarrow q,$
$\qquad SplitValue \leftarrow p, Size \leftarrow |\mathcal{D}|)$

$\bar{f}_{\mathbf{m}}(\mathbf{x})$: $E[\bar{f}_{\mathbf{m}}(\mathbf{x})]$ as follows [17].

$$\begin{aligned} MSE(\bar{f}_{\mathbf{m}}(\mathbf{x})) &= \big\{E[\bar{f}_{\mathbf{m}}(\mathbf{x})] - p_d(\mathbf{x})\big\}^2 \\ &\quad + E\big[\{\bar{f}_{\mathbf{m}}(\mathbf{x}) - E[\bar{f}_{\mathbf{m}}(\mathbf{x})]\}^2\big]. \end{aligned}$$

The first term on the rhs is called "*square bias* " and the second "*variance*." We evaluate the magnitude of each of these two terms in the following.

To simplify notations for the rest of the paper, we have used $T_i(\mathbf{x})$ to denote $T_i(\mathbf{x}|\mathcal{D}_i)$, and $p(T_i(x))$ to denote $p(\mathbf{x}_k \in T_i(\mathbf{x})|\ \mathbf{x}_k \in \mathcal{D}_i)$.

Let $\mathbf{c}_i$ be the center of a region of $T_i(\mathbf{x})$ where each element c_{ij} of $\mathbf{c}_i$ is a middle point of the interval on each dimension j. The second order Taylor approximation of $p_d(\mathbf{x})$ around $\mathbf{c}_i$ for $T_i(\mathbf{x})$ is given as

$$\begin{aligned} p_d(\mathbf{x})|_{\mathbf{c}_i \in T_i(\mathbf{x})} &\approx p_d(\mathbf{c}_i) + (\mathbf{x} - \mathbf{c}_i)^\mathsf{T} \nabla p_d(\mathbf{x})|_{\mathbf{x}=\mathbf{c}_i} \\ &\quad + \frac{1}{2}\{(\mathbf{x} - \mathbf{c}_i)^\mathsf{T} \nabla\}^2 p_d(\mathbf{x})|_{\mathbf{x}=\mathbf{c}_i}, \quad (3) \end{aligned}$$

where $\nabla = [\partial/\partial x_1, \ldots, \partial/\partial x_d]^\mathsf{T}$.

Note that $\mathbf{m}(T_i(\mathbf{x}))$ follows a binomial distribution[2] $B(\psi, p(T_i(x)))$. Therefore, $E[\bar{f}_{\mathbf{m}}(\mathbf{x})]$ is expressed by substituting $E[\mathbf{m}(T_i(\mathbf{x}))] = \psi p(T_i(x))$ in Eq. (2).

$$\begin{aligned} E[\bar{f}_{\mathbf{m}}(\mathbf{x})] &= \frac{1}{t}\sum_{i=1}^{t}\frac{E[\mathbf{m}(T_i(\mathbf{x}))]}{\psi v_i} \\ &= \frac{1}{t}\sum_{i=1}^{t}\frac{p(T_i(x))}{v_i} \\ &= \frac{1}{t}\sum_{i=1}^{t}\frac{1}{v_i}\int_{T_i(\mathbf{x})} p_d(\mathbf{x}_*)d\mathbf{x}_*. \end{aligned} \quad (4)$$

Accordingly, the square bias is evaluated as follows by applying Eq. (3) and the fact that the integral of an odd function over $[c_{ij} - \delta x_j/2, c_{ij} + \delta x_j/2]$ for each dimension j is zero.

$$\begin{aligned} &\left\{E[\bar{f}_{\mathbf{m}}(\mathbf{x})] - p_d(\mathbf{x})\right\}^2 \\ &\approx \Bigg[\frac{1}{t}\sum_{i=1}^{t}\Bigg\{\frac{1}{24}\sum_{j=1}^{d}\frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\Big|_{\mathbf{x}=\mathbf{c}_i}\delta x_{ij}^2 \\ &\quad -(\mathbf{x}-\mathbf{c}_i)^{\mathsf{T}}\nabla p_d(\mathbf{x})|_{\mathbf{x}=\mathbf{c}_i} \\ &\quad -\frac{1}{2}\{(\mathbf{x}-\mathbf{c}_i)^{\mathsf{T}}\nabla\}^2 p_d(\mathbf{x})|_{\mathbf{x}=\mathbf{c}_i}\Bigg\}_{\mathbf{c}_i\in T_i(\mathbf{x})}\Bigg]^2 \\ &\leq \Bigg[\frac{1}{t}\sum_{i=1}^{t}\Bigg\{\frac{1}{24}\Bigg|\sum_{j=1}^{d}\frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\Big|_{\mathbf{x}=\mathbf{c}_i}\Bigg|\Delta_{ij}^2 2^{-2h} \\ &\quad +\sum_{j=1}^{d}\Bigg|\frac{\partial p_d(\mathbf{x})}{\partial x_j}\Big|_{\mathbf{x}=\mathbf{c}_i}\Bigg|\Delta_{ij}2^{-h} \\ &\quad +\frac{1}{2}\sum_{j=1}^{d}\sum_{k=1}^{d}\Bigg|\frac{\partial^2 p_d(\mathbf{x})}{\partial x_j\partial x_k}\Big|_{\mathbf{x}=\mathbf{c}_i}\Bigg|\Delta_{ij}\Delta_{ik}2^{-2h}\Bigg\}_{\mathbf{c}_i\in T_i(\mathbf{x})}\Bigg]^2 \\ &\quad = O(4^{-h}) \end{aligned}$$

This result shows that the square bias diminishes as level h increases, *i.e.*, as the size of the regions decreases. Though this analysis uses the second order approximation of $p_d(\mathbf{x})$, the result using the higher order approximation is the same since the first order term dominates in the above formula.

Because $\mathbf{m}(T_i(\mathbf{x}))$ follows the binomial distribution $B(\psi, p(T_i(x)))$, the variance of $\mathbf{m}(T_i(\mathbf{x}))$ is

$$var[\mathbf{m}(T_i(\mathbf{x}))] = \psi p(T_i(x))(1 - p(T_i(x))).$$

In concert with Eq. (2), the variance of $\bar{f}_{\mathbf{m}}(\mathbf{x})$ is represented as follows.

$$\begin{aligned} &E\left[\{\bar{f}_{\mathbf{m}}(\mathbf{x}) - E[\bar{f}_{\mathbf{m}}(\mathbf{x})]\}^2\right] \\ &= \frac{1}{t^2}\sum_{i=1}^{t}\frac{p(T_i(x))(1-p(T_i(x)))}{\psi v_i^2} \\ &= \frac{1}{t^2}\sum_{i=1}^{t}\frac{1}{\psi v_i^2}\int_{T_i(\mathbf{x})} p_d(\mathbf{x}_*)d\mathbf{x}_*\left(1 - \int_{T_i(\mathbf{x})} p_d(\mathbf{x}_*)d\mathbf{x}_*\right). \end{aligned}$$

Using the similar calculus as applied to the square bias, we obtain the variance as follows where $\mathbf{c_i}$ is a center of $T_i(\mathbf{x})$.

$$\begin{aligned} &E\left[\{\bar{f}_{\mathbf{m}}(\mathbf{x}) - E[\bar{f}_{\mathbf{m}}(\mathbf{x})]\}^2\right] \\ &\approx \frac{1}{t^2}\sum_{i=1}^{t}\frac{1}{\psi}\left\{p_d(\mathbf{c}_i) + \frac{1}{24}\sum_{j=1}^{d}\frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\Big|_{\mathbf{x}=\mathbf{c}_i}\delta x_{ij}^2\right\} \\ &\quad \times\left\{\frac{1}{v_i} - p_d(\mathbf{c}_i) - \frac{1}{24}\sum_{j=1}^{d}\frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\Big|_{\mathbf{x}=\mathbf{c}_i}\delta x_{ij}^2\right\}. \\ &= \frac{1}{t^2}\sum_{i=1}^{t}\frac{1}{\psi}\left\{p_d(\mathbf{c}_i) + \frac{1}{24}\sum_{j=1}^{d}\frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\Big|_{\mathbf{x}=\mathbf{c}_i}\Delta_{ij}^2 2^{-2h}\right\} \\ &\quad \times\left\{\frac{2^{dh}}{\prod_{j=1}^{d}\Delta_{ij}} - p_d(\mathbf{c}_i) - \frac{1}{24}\sum_{j=1}^{d}\frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\Big|_{\mathbf{x}=\mathbf{c}_i}\Delta_{ij}^2 2^{-2h}\right\}. \\ &\quad = O(2^{dh}) \end{aligned}$$

This result indicates that the variance increases when level h increases. Also, the result does not change even if we use the higher order approximation because the term $p_d(\mathbf{c}_i)/v_i$ dominates in the above formula.

The property of DEMass, revealed from this error analysis, is similar to that of the conventional kernel density estimator which shows a bias-variance trade off—the bias decreases as the kernel bandwidth b decreases but this increases the variance; and the reverse is true if the kernel bandwidth in increased [17]. The parameter k in k-NN density estimator has the same effect.

In conclusion, DEMass has a comparable estimation of density with the kernel density estimator if both trade-off bias and variance equally well; and it is indeed the case in practice. Figure 1 shows the estimation result of a normal distribution using KDE and DEMass, respectively. It demonstrates that DEMass produces similar result to that generated by KDE, for different data sizes. Smoothing can be applied by increasing b for KDE or decreasing h for DEMass which produces the estimation results as shown in Figure 2. The parameters used for DEMass are: $t = 1000$ and $\psi = n$ when $n = 10, 100$; $\psi = 1000$ when $n = 1000000$.

Note that in either settings shown in Figures 1 and 2, the estimations of both KDE and DEMass approach the true distribution as the number of instances increases.

[2]The implementation of $T(\cdot)$ used in this paper is a tree-based nonparametric method. The binomial distribution is required for the error analysis only.

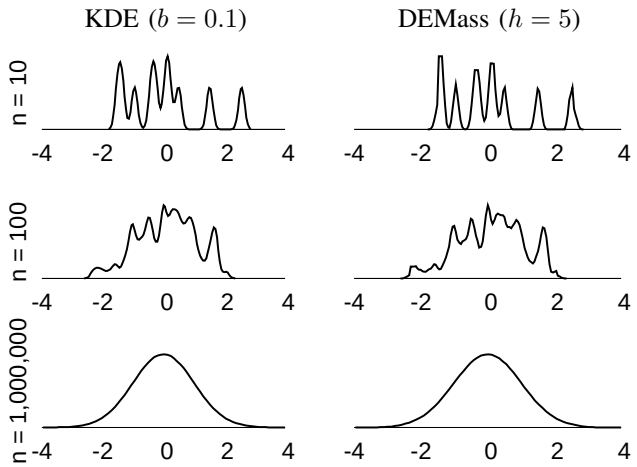


Figure 1: Example estimations of Kernel Density Estimator (with Gaussian kernel) using $b = 0.1$ and DEMass using $h = 5$ for different data sizes, $n = 10, 100, 1000000$. The true data distribution is a normal distribution.

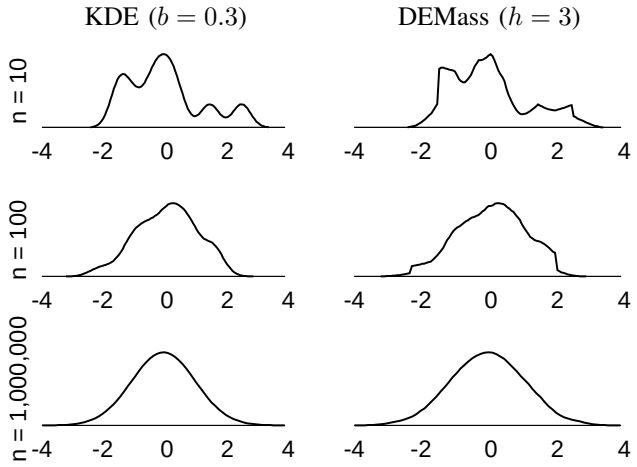


Figure 2: Example estimations of Kernel Density Estimator (with Gaussian kernel) using $b = 0.3$ and DEMass using $h = 3$ for the same data used in Figure 1.

V. Using DEMass in existing density-based algorithms

This section describes how DEMass can be applied to three current density-based algorithms, DBSCAN [7], LOF [5] and Bayesian classifier, in place of their existing density estimators. DBSCAN, LOF and Bayesian classifier, are one of the best algorithms for clustering, anomaly detection and classification, respectively.

Using DEMass automatically carries the two advantages mentioned in Section III: (i) The estimation requires no distance measures, thus, it completely saves the cost of distance calculations for every pair of instances; and (ii) DEMass enables small samples to construct the required regions $T(\cdot|\mathcal{D})$, overcoming the key limitation of DBSCAN and LOF in handling very large databases. We will discuss further advantages specific to individual algorithms in the following subsections.

A. DEMass-DBSCAN

The principal steps of DEMass-DBSCAN is the same as DBSCAN, except that no border points and their associated step are required. A comparison of the two algorithms are provided in Table I. The algorithm for DBSCAN is adapted from [18].

step	DBSCAN	DEMass-DBSCAN
1	Label all points as core, border, or noise points, based on $\bar{f}_{kNN}(\mathbf{x})$	Label all $T(\mathbf{x})$ satisfying Definition 1 as core regions, based on $\bar{f}_{\mathbf{m}}(\mathbf{x})$. Points not covered by core regions are noise.
2	Eliminate noise points	Eliminate noise points
3	Connect all core points that are within ϵ of each other.	Connect all core regions that have non-zero intersections.
4	Make each group of connected core points into a separate cluster	Make each group of connected core regions into a separate cluster.
5	Assign each border point to one of the clusters of its associated core points.	

Table I: Algorithms for DBSCAN and DEMass-DBSCAN. Note that border points are not required with DEMass-DBSCAN; thus step 5 is not needed. Both versions of DBSCAN could include an additional cluster size threshold to eliminate small size clusters in the last step.

While following its principal steps, the use of DEMass simplifies DBSCAN in two ways, in addition to the two advantages already mentioned above. First, DEMass enables regions to be labelled instead of individual points. Because the number of regions is significantly less than the number of points, linking regions to form a cluster becomes significantly faster than connecting points. Second, no border points need to be defined because the connections within a cluster are established via core regions only when DEMass is used. The first simplification is the key reason for the significant speed up achieved by DEMass-DBSCAN, which we will show in Section VI-A.

The DEMass-DBSCAN algorithm shown in Table I is derived from set-based definitions; and the DBSCAN algorithm is derived from point-based definitions. We can use point-based definitions for DEMass-DBSCAN, except that the neighbourhood definition needs to be adapted to DEMass. Although point-based definitions can be defined as in DBSCAN [7], set-based definitions are simpler. The formal point-based and set-based definitions for DEMass-DBSCAN are given in table II.

Point-based definitions	Set-based definitions
Definition P1: The h-**neighbourhood** of a point p, denoted by $N_h(p)$, is defined by $N_h(p) = \{q \in D | q \in T^h(p)\}$. In contrast, the ϵ-neighbourhood for DBSCAN is defined as $N_\epsilon(p) = \{q \in D | dist(p,q) \leq \epsilon\}$, which requires a distance function $dist(\cdot,\cdot)$. No distance functions are required for $N_h(\cdot)$.	
Definition P2: A point p is **directly density-reachable** from a point q wrt h and $MinPts$ if (i) $p \in N_h(q)$ and (ii) $\lvert N_h(q)\rvert \geq MinPts$ (core point condition).	*Definition S1:* $T(\mathbf{x})$ is a **core region** of point $\mathbf{x}$ wrt h and $MinPts$ if $\mathbf{m}(T(\mathbf{x}))\frac{v_{max}}{v} \geq MinPts$. where $v_{max} = \max_i v_i$, and v is the volume of region $T(\mathbf{x})$. The normalisation factor $\frac{v}{v_{max}}$ ranges from 0 to 1.
Definition P3: A point p is **density-reachable** from a point q wrt h and $MinPts$ if there is a chain of points $p_1, \ldots, p_n$, where $p_1 = p$ and $p_n = q$ such that p_{i+1} is directly density-reachable from p_i.	
Definition P4: A point p is **density-connected** to a point q wrt h and $MinPts$ if there is a point o such that both p and q are density-reachable from o wrt h and $MinPts$.	*Definition S2:* $T_r(\cdot)$ is **density-connected** to $T_s(\cdot)$ wrt h and $MinPts$ if there is a chain of regions $T_1(\cdot), \ldots, T_g(\cdot)$ where $r = 1$ and $s = g$ such that $T_i(\cdot) \cap T_{i+1}(\cdot) \neq \emptyset$ and $T_i(\cdot)$ is a core region for all i wrt h and $MinPts$.
Definition P5: Let D be a database of points. A **cluster** C wrt h and $MinPts$ is a non-empty subset of D satisfying the following conditions: (i) $\forall p, q$: if $p \in C$ and q is density-reachable from p wrt h and $MinPts$, then $q \in C$. (Maximality) (ii) $\forall p, q \in C$: p is density-connected to q wrt h and $MinPts$. (Connectivity)	*Definition S3:* An **arbitrary-shape cluster** C wrt h and $MinPts$ is a non-empty subset of a database D satisfying the following condition: $\forall r, s; T_r(\cdot), T_s(\cdot) \subset C$: $T_r(\cdot)$ is density-connected to $T_s(\cdot)$ wrt h and $MinPts$.
Definition P6: Let $C_1, \ldots, C_k$ be the clusters of the database D wrt h and $MinPts$. Then we define **noise** as the set of points in the database D not belonging to any cluster C_j, i.e., noise = $\{p \in D | \forall j : p \notin C_j\}$.	*Definition S4:* Let $C_1, \ldots, C_k$ be the clusters of D wrt h and $MinPts$. **Noise** is the set of points in D not belonging to any cluster C_j, i.e., noise = $\{\mathbf{x} \in D | \forall j : \mathbf{x} \notin C_j\}$.

Table II: Point-based and set-based definitions for DEMass-DBSCAN. Note that the point-based definitions are adopted from those defined for DBSCAN [7].

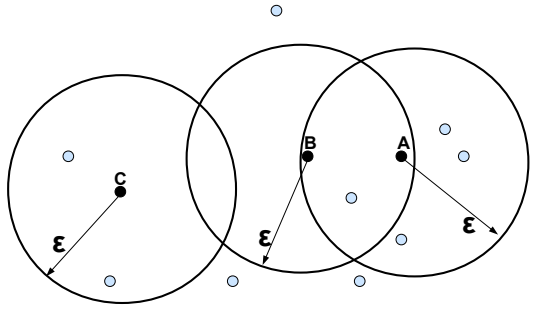


Figure 3: An example for DBSCAN for $Minpts = 5$. A is a core point, B is a border point, and C is a noise point.

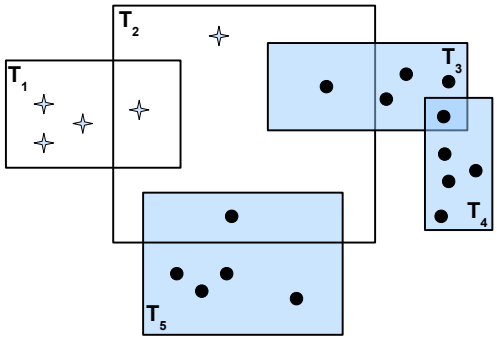


Figure 4: An example for DEMass-DBSCAN for $Minpts = 5$. The circle symbol indicates core points and the star symbol indicates noise points. T_3, T_4, and T_5 are core regions. T_3 and T_4 are linked by a common core point.

A comparison between DBSCAN and DEMass-DBSCAN is provided using two examples showed in Figures 3 and 4. They show how core points and non-core points are labelled in DBSCAN and DEMass-DBSCAN. One superficial difference in these examples is that DBSCAN uses hyper-spheres and DEMass-DBSCAN uses hyper-rectangles. This difference can be easily eliminated by using L^∞-norm (instead of L^2-norm) in DBSCAN.

B. DEMass-LOF

Table III compares the algorithms for LOF and DEMass-LOF which have three identical principal steps: Compute density distribution and LOF, and then rank all instances based on their LOF values. The key difference is the density estimator used in step 1 which changes the computation of LOF in step 2.

In addition to the two advantages due to the use of DEMass mentioned in Section III, the advantage specific to LOF is that DEMass enables the computation of the relative density to be substantially simplified, changing from nearest-neighbour-based to set-based. Instead of finding the neighbours of $\mathbf{x}$ and then compute the density of each neighbours, the modified ranking measure LOF_p is computed based on the region $T(\mathbf{x})$ and its immediate larger region $\breve{T}_i(\mathbf{x}) \supset T_i(\mathbf{x})$. In the tree implementation of $T(\cdot)$, this corresponds to computing the density of the node in which x falls into, relative to the density of its parent node.

In steps 1 and 2, the time complexities of DEMass-LOF and LOF are $O(nt\psi)$ and $O(n^2)$, respectively. Since DEMass-LOF does not need to perform neighbourhood

step	LOF	DEMass-LOF
1	Compute density distribution: $\bar{f}_{kNN}(\mathbf{x})$	Compute density distribution: $\bar{f}_{\mathbf{m}}(\mathbf{x})$
2	Compute $LOF(\mathbf{x})$ using $\frac{\sum_{\mathbf{x}' \in N(\mathbf{x},k)} \frac{\bar{f}_{kNN}(\mathbf{x}')}{\lvert N(\mathbf{x},k)\rvert}}{\bar{f}_{kNN}(\mathbf{x})}$	Compute $LOF_p(\mathbf{x})$ using: $\frac{\frac{1}{t}\sum_{i=1}^{t} \frac{\mathbf{m}(\breve{T}_i(\mathbf{x}))}{\breve{\psi}\breve{v}_i}}{\bar{f}_{\mathbf{m}}(\mathbf{x})}$
3	Rank all instances based on their LOF values in descending order	Rank all instances based on their LOF_p values in descending order

Table III: Algorithms for LOF and DEMass-LOF. $\bar{f}_{kNN}(\mathbf{x})$ and $N(\mathbf{x}, k)$ are defined in Section II-B; $\bar{f}_{\mathbf{m}}(\mathbf{x})$ is defined in Section III. $\breve{T}_i(\mathbf{x}) \supset T_i(\mathbf{x})$ correspond to the parent and child nodes in our tree implementation; $\breve{\psi}$ and $\breve{v}_i$ are the data size and volume of $\breve{T}_i(\mathbf{x})$, respectively. Note that $\breve{T}_i(\mathbf{x})$, the next superset of $T_i^h(\mathbf{x})$, is not necessarily $T_i^{h-1}(\mathbf{x})$ because there are d levels in the tree for each increment of h and the implementation allows single branch extensions if there are no data in other branches. See III-B for the details of the implementation.

search as in LOF, it is much faster, especially in large data sets. This is because DEMass-LOF does not need to compute the density of all neighbours of each instance.

The parameter k in LOF has an inverse relationship with h in DEMass-LOF, i.e., high h corresponds to low k (which covers a smaller region than that using low h or high k).

A larger k increases LOF's processing time so as a larger h increases DEMass-LOF's processing time.

Note that both LOF and LOF_p are relative density scores, which range from 0 to $+\infty$, indicating the degree of anomaly; the higher the score, the higher the degree of anomaly.

C. DEMass-Bayes

Bayesian classifiers require density estimation in order to estimate the class conditional probability. Classifiers based on Bayes rule estimate the likelihood of test instance $\mathbf{x}$ given class y i.e. $p(\mathbf{x}|y)$. Because it has been difficult to compute $p(\mathbf{x}|y)$ directly even in problems with a moderate number of dimensions, a number of assumptions has been made to simplify the computation. We describe those used in Naive Bayes (NB) [12], Bayesian Networks (BayesNet) [9], and Aggregating One-Dependence Estimator (AODE) [23], in the following two paragraphs.

Naive Bayes assumes class conditional independence and estimates density distribution on each dimension separately [12].

$$p(\mathbf{x}|y) = \prod_{i=1}^{d} p(x_i|y) \tag{5}$$

The assumption made by Naive Bayes is often violated in the real world where attributes are related in some way. Other Bayesian classifiers such as BayesNet [9] and AODE [23], employ less restrictive assumptions. BayesNet learns probabilistic relationships among the attributes in the form of directed acyclic graph (DAG) from the training data. In a graph, edges represent conditional dependencies and nodes which are not connected are conditionally independent. At each node, joint probabilities with respect to its parents are learned from the training data. AODE allows conditional dependence with one 'privileged' attribute. Other attributes are conditionally independent given the class label y and a privileged attribute x_i and conditional probabilities are computed as follows.

$$p(\mathbf{x}|x_i, y) = \prod_{j=1}^{d} p(x_j|x_i, y) \tag{6}$$

As BayesNet and AODE can not handle numeric attributes, they discretise numeric attributes and compute the conditional probabilities. For numeric attributes, $p(x_i|y)$ in Naive Bayes can either be estimated by Gaussian distribution (through normal distribution assumption)(NB-GD) [12] or by kernel density estimation (NB-KDE) [13].

In contrast to the existing implementation of Bayesian classifiers, the implementation based on DEMass estimates $p(\mathbf{x}|y)$ directly, without any assumptions. In order to use DEMass in classification, we made the following adjustments to estimate $p(\mathbf{x}|y)$.

- Instead of randomly selecting ψ samples from the training data to construct a tree, ψ samples are selected from and a tree is built separately for each class. If there are not enough data to sample in a class, all data from that class are used i.e. $\psi_y = min(\psi, |\mathcal{C}_y|)$, where $\mathcal{C}_y$ is the set of instances belonging to class y. We build t trees per class and hence, $c \times t$ trees are constructed in total, where c is the number of classes.
- Instead of growing the tree to the maximum height, we stop growing when the number of instances in a node is less than or equal to one. This provides a smoother estimation.

We compute the class conditional probability based on DEMass as:

$$p(\mathbf{x}|y) \equiv \bar{f}_{\mathbf{m}}(\mathbf{x}|y) = \frac{1}{t}\sum_{i=1}^{t} \frac{\mathbf{m}(T_i(\mathbf{x}|\mathcal{D}_{i,y}))}{\psi v_i} \tag{7}$$

where $\mathcal{D}_{i,y}$ is a subset of ψ samples from class y. The rest of the steps in DEMass-Bayes are the same as existing Bayesian classifiers. The prior probabilities $p(y)$ are calculated from the training data. Finally, Bayes rule is used to predict the class which has the maximum posterior $p(y|\mathbf{x})$.

$$\hat{y} = \arg\max_{y}(p(\mathbf{x}|y)p(y)) \tag{8}$$

The decision rules of existing Bayesian classifiers and DEMass-Bayes are provided in Table IV.

Classifier	Decision Rule	Remarks
NB-GD	$\arg\max_y(p(\mathbf{x}\|y)p(y))$	$p(x_i\|y)$ in eq.5 is estimated with norm. dist.
NB-KDE		$p(x_i\|y)$ in eq.5 is estimated with KDE.
DEMass-Bayes		$p(\mathbf{x}\|y)$ is estimated using eq.7.
Bayes Net	$\arg\max_y(\prod_i^d p(x_i\|\pi_i,y)p(\pi_i,y))$	$\pi_i = parents(x_i)$; Joint probabilities are estimated by discretisation.
AODE	$\arg\max_y(\sum_{i=1}^d p(\mathbf{x}\|x_i,y)p(x_i,y))$	$p(x_j\|x_i,y)$ in eq.6 is estimated by discretisation.

Table IV: Decision rules of different existing Bayesian classifiers and DEMass-Bayes.

VI. Empirical evaluation

The evaluations in clustering and anomaly detection tasks, are conducted in the unsupervised learning setting, whereas classification task in the supervised setting. We will compare DBSCAN with DEMass-DBSCAN in the first subsection and then compare LOF with DEMass-LOF in the second subsection. Finally, we will compare DEMass-Bayes with four existing Bayesian classifiers namely, NB-GD, NB-KDE, BayesNet and AODE in the last subsection.

All experiments were conducted as single thread jobs processed at 2.3 GHz in a Linux cluster (www.vpac.org) using a node with 32 GB memory. All DEMass based algorithms were written in JAVA in WEKA platform [24], so as DBSCAN and existing Bayesian classifiers. LOF was written in Java in ELKI platform version 0.4 [1]. The data sets used are from UCI Machine Learning Repository [2], unless stated otherwise.

The clustering result was reported in terms of CPU runtime (in seconds), number of clusters identified, number of unassigned instances, and F-measure which was calculated based on assigned instances only. F-measure = 1 when all assigned instances are in the correct clusters, i.e., perfect clustering; and F-measure = 0 if all instances are assigned to wrong clusters. The anomaly detection result was reported in terms of CPU runtime and AUC (Area Under ROC Curve) based on the ranked result. The classification result was reported in terms of classification accuracy and CPU runtime (in seconds). We tuned the parameters of each algorithm in the unsupervised learning setting and reported the best result. In the supervised learning, the default parameter settings were used for all the classifiers; and the reported results were from a 10-fold cross validation.

A. DEMass-DBSCAN versus DBSCAN

DEMass-DBSCAN had $\psi = 256$ and $t = 1000$ as default; and both DEMass-DBSCAN and DBSCAN used $MinPts = 6$ in all experiments. As a result, only one parameter needed to be tuned for a particular data set: h for DEMass-DBSCAN and ϵ for DBSCAN.

Table V: Clustering results in the Ring-Curve+Wave+Tri-Gaussian data sets for DEMass-DBSCAN ($h = 7$ for 3-dimensional data; $h = 6$ for 48-dimensional data) and DBSCAN ($\epsilon = 0.01$).

	3-dimensional data		48-dimensional data	
	DEMass-DBSCAN	DBSCAN	DEMass-DBSCAN	DBSCAN
Runtime	135	2391	1261	21906
#cluster [7]	9	8	7	8
#unassigned	535	332	61	332
F-measure	0.9999	0.9999	1.0000	0.9999

Ring-Curve-Wave-Tri-Gaussian. It has three two-dimensional synthetic data embedded in either a 3-dimensional data set or a 48-dimensional data set (where 42 dimensions are irrelevant with a constant value). The three two-dimensional data are Ring-Curve, Wave and Triangular-Gaussian shown in Figure 10 in Appendix, which have a total of seven clusters. Each cluster has 10000 instances with a total of 70000 instances.

The clustering results from DEMass-DBSCAN and DBSCAN are shown in Table V. DEMass-DBSCAN ran faster than DBSCAN by a factor more than 17 in both data sets. In terms of #clusters and #unassigned, DEMass-DBSCAN performed slightly worse than DBSCAN in the 3-dimensional data set, but better in the 48-dimensional data set. DEMass-DBSCAN decreased its number of unassigned instances from 535 to 61 when the number of dimensions was increased from 3 to 48; whereas DBSCAN had the same 332 unassigned instances in both cases. DEMass-DBSCAN performs either similarly to or better than DBSCAN in terms of F-measure in these two data sets.

In order to examine how well the algorithms scale up to large data size, we used the 48-dimensional data set and increased the data size from 7000 to 70000, half-a-million, 1 million and 10 million. Figure 5 plotted runtime ratio versus data size ratio (1, 10, 75, 150 and 1500) by using 7000 as the base. The result showed that DEMass-DBSCAN had a sublinear increase in runtime: The runtime ratio increased from 1 to 101 when the data size ratio increased from 1 to 150. In contrast, DBSCAN's runtime ratio increased from 1 to 18000 with the same increase in data size ratio. DEMass-DBSCAN was faster than DBSCAN by a factor of 193 when the one-million data set is used. Even the data size was increased by a factor of 1500, the runtime of DEMass-DBSCAN increased by a factor of 862 only.

OneBig and Pendigits. The OneBig data set [15] has 20 attributes, 9 clusters and a total of 68000 instances. The biggest cluster has 50011 instances, and each of the other eight clusters has approximately 1000 instances. In addition, there are 10000 noise instances randomly distributed in the feature space. The Pendigits data set has 16 attributes and

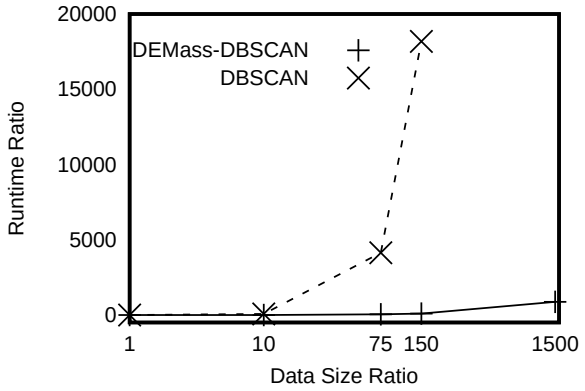


Figure 5: Scale up: DEMass-DBSCAN vs DBSCAN in the 48-dimensional Ring-Curve-Wave-TriGaussian data set. Note that DBSCAN completed the task of the one-million data set (at data size ratio=150) in 36 days versus DEMass-DBSCAN's 4.5 hours. Even with the 10-million data set, DEMass-DBSCAN completed it in 38 hours.

Table VI: Clustering results in the OneBig and Pendigits data sets for DEMass-DBSCAN ($h = 3$ for OneBig; $h = 2$ for Pendigits) and DBSCAN ($\epsilon = 0.1$ for OneBig; $\epsilon = 0.2$ for Pendigits).

	OneBig		Pendigits	
	DEMass-DBSCAN	DBSCAN	DEMass-DBSCAN	DBSCAN
Runtime	1145	8544	91	204
#cluster	[9] 9	9	[10] 47	65
#unassigned	10021	10005	2166	6251
F-measure	1.00	1.00	0.65	0.75

10 clusters. Each cluster has approximately 1100 instances which makes up a total of 10992 instances.

The result in Table VI showed that DEMass-DBSCAN and DBSCAN for OneBig had the same clustering result in terms of F-measure and number of clusters; but DEMass-DBSCAN ran faster than DBSCAN by a factor of 7. Note that DEMass-DBSCAN had correctly identified all but one of the 10000 noise instances; whereas DBSCAN correctly identified all of the noise instances. For Pendigits, the result showed that although DEMass-DBSCAN had a lower F-Measure than DBSCAN, it was better than DBSCAN in all other measures: it had only 20% instances unassigned whereas DBSCAN had 57% instances unassigned; DEMass-DBSCAN found 47 cluster whereas DBSCAN detected 65.

B. DEMass-LOF versus LOF

For anomaly detection tasks, we compared LOF with DEMass-LOF in this section. Table VII provided the properties of the data sets used. Note that Http and Smtp are subsets of the network intrusion data set used in KDDCUP 99 [25]; and an anomaly data generator, "Mulcross" [16] is used to generate a synthetic data set. All the data sets used have nearly fifty thousand or more instances, with the largest up to half a million instances. The default settings for DEMass-LOF were $\psi = 256$ and $t = 100$.

Table VII: Data sets used for the anomaly detection task for comparing DEMass-LOF with LOF.

Data	Size n	d	anomaly class
Http	567497	3	attack (0.4%)
ForestCover (FC)	286048	10	class 4 (0.9%) vs. class 2
Mulcross	262144	4	2 clusters (10%)
Smtp	95156	3	attack (0.03%)
Shuttle	49097	8	classes 2,3,5,6,7 (7%)

Table VIII: Compare LOF and DEMass-LOF in terms of AUC (Area Under ROC Curve) and time (in seconds). AUC=1 is the perfect detection performance and AUC=0 is the worst. The default settings for DEMass-LOF were $h = 1$, $\psi = 256$ and $t = 100$ which were used for all data sets. The parameter k (for LOF) and h (for DEMass-LOF) were changed in order to explore a better result.

	AUC				Time (seconds)			
	LOF		DEMass-LOF		LOF		DEMass-LOF	
	k=10	k=60	h=1	h=4	k=10	k=60	h=1	h=4
Http	0.44	0.35	0.99	0.93	18913	19818	19	42
FC	0.57	0.58	0.74	0.77	10835	11147	39	40
Mulcross	0.59	0.59	0.96	0.09	5432	5486	12	53
Smtp	0.32	0.85	0.29	0.89	540	552	2	5
Shuttle	0.55	0.62	0.94	0.71	368	380	5	12

Table VIII compares LOF with DEMass-LOF in terms of detection performance AUC and time. DEMass-LOF using either h=1 or 4 obtained better AUC results than LOF. It is interesting to note that DEMass-LOF achieved extreme results in the Smtp and Mulcross data sets between the two h settings; and it behaved differently in these two data sets, where a low h setting is better in Mulcross but a high h setting is better in Smtp. This is because the two data sets have two different types of anomalies: clustered and scattered anomalies [14]. Mulcross has clustered anomalies, i.e., outlying clusters with high density but a small number of instances. DEMass-LOF with a high h setting (i.e., h=4) regarded these anomaly clusters more 'normal' than normal instances, which was reflected in the result: AUC=0.09. In contrast, the Smtp data set has scattered anomalies which are isolated outlying instances around normal clusters. This scenario requires a high h setting in order for DEMass-LOF to compute the right densities for these anomalies.

LOF was not competitive, and the AUC results did not change much from the presented results even other k values were used (we had tried k=30, 40, 50, 80, 100, 120.)

However, it shall be noted that LOF could achieve good detection accuracy with an appropriate k. For example, LOF obtained AUC=0.99 when $k = 4000$ was used in the shuttle data set. But similar search in the largest three data sets failed with out of memory problem even though the computer system was allocated 32 GB memory! This result reveals two universal problems with k-NN approaches like

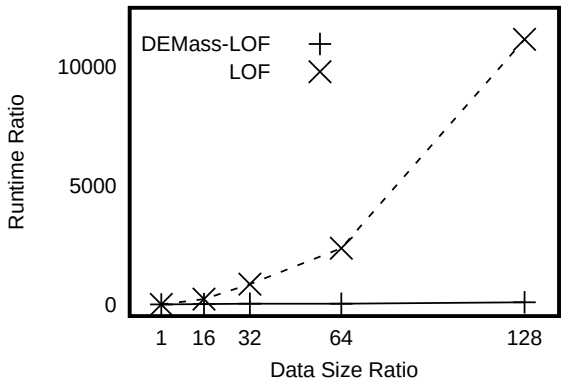


Figure 6: Scale up: LOF versus DEMass-LOF in Mulcross. The base for data size ratio is 8192 instances and the base for runtime ratio is the runtime on 8192 instances.

LOF: (i) An extensive parameter search is required to obtain good detection accuracy; this search adds a significant cost to the already long runtime process. The total time cost is often prohibitive; and (ii) high memory requirement.

Table VIII also compares these detectors in terms of processing time. DEMass-LOF was one to near-three orders of magnitude faster than LOF in these data sets.

Figure 6 showed the runtime of both algorithms when scaling from 8192 instances up to a million instances in the Mulcross data set. The data size was increased by a factor of 16, 32, 64, 128 from 8192 instances. DEMass-LOF increased its runtime by a factor of 11, 23, 25 and 86, respectively. In contrast, LOF increased its runtime by a factor of 217, 845, 2371, 11173, respectively. At data size ratio = 128, which has a million instances, LOF completed the task in 28 hours whereas DEMass-LOF accomplished it in 45 seconds!

C. DEMass-Bayes versus Bayesian classifiers

In this subsection, we compare the performance of DEMass-Bayes with four existing Bayesian classifiers: NB-GD, NB-KDE, BayesNet and AODE.

For better estimation of multidimensional density, we need sufficient training data. Hence, we chose bigger data sets with size $n > 10000$. We tested on 10 data sets with different sizes, dimensions, number of classes and class distributions. The properties of the data sets are provided in Table IX. Out of 10 data set used, Wave, RingCurve, OneBig and Mulcross are synthetic and the rest are real data sets. RingCurve and Wave are the subsets of Ring-Curve-Wave-Tri-Gaussian data set described in section VI-A, each having two classes with 10000 data points in each class. OneBig and Pendigits are the same data sets as discussed in section VI-A. In OneBig, noise in the data set are treated as a separate class, hence, it has 10 classes. Mulcross is a synthetic data set used in section VI-B. It has two classes with 235930 and 26214 instances. Magic04 has two classes with 12332 and 6688 instances. Letter Recognition (LetRecog) is a data set of 26 characters with approximately 750 data instances in each class. Out of 7 classes in Shuttle, approximately 80% of the data belongs to the first class whereas the smallest class has 10 instances only. MiniBooNE is a dataset from an experiment to distinguish electron neutrinos (signal) from muon neutrinos (background). The class distribution is approximately 7:3. CoverType is a data set of forest cover type with 7 classes. It is the biggest data set used with more than half a million data. The class distribution is unbalanced as two classes have more than two hundred thousand instances each and the smallest class has 2747 instances.

Table IX: Data set used in classification task to compare the performance of DEMass-Bayes with other existing Bayesian classifiers.

Data set	Size(n)	Attributes#	Classes
Pendigits	10992	16	10
Magic04	19020	10	2
Wave	20000	2	2
RingCurve	20000	2	2
LetRecog	20000	16	26
Shuttle	58000	8	7
OneBig	68000	20	10
MiniBooNE	129596	50	2
Mulcross	262144	4	2
CoverType	581012	10	7

The default setting for DEMass-Bayes were $t = 1000$, $\psi = 1024$ and h was set as:

$$h = \begin{cases} \lceil \log_2(\psi) \rceil & \text{if } d = 1 \\ \lceil \log_d(\psi) \rceil & \text{otherwise.} \end{cases}$$

We normalised the data in the range of $[0 - 1]$ to avoid attributes with large values affecting volume calculation.

Since AODE can not handle continuous attributes, we discretised the attributes using the method proposed in [8]. BayesNet does discretization before building the classification model.

We performed 10-fold cross validation and reported the average accuracy and total run time including training and testing over all 10 folds. The classification accuracies and runtime (in seconds) of all classifiers over a 10-fold cross validation on each data set are provided in Table X and Table XII respectively.

Empirical observations showed that DEMass-Bayes yielded better classification accuracies in most of the data sets. DEMass-Bayes outperformed existing Bayesian classifiers in 6 data sets namely Pendigits, Wave, RingCurve, Letter Recognition, OneBig, and Covertype. In case of Wave, Covertype and Letter Recognition, DEMass-Bayes had significant difference in accuracy over existing Bayesian classifiers with improvement of 20%, 9% and 6% respectively. It had slightly poorer accuracy than the existing

Table X: Classification accuracies (%) on different data sets over 10-fold cross validation for DEMass-Bayes and existing Bayesian classifiers: NB-KDE, NB-GD, BayesNet and AODE with default parameters.

Data set	DEMass -Bayes	NB-KDE	NB-GD	Bayes Net	AODE
Pendigits	**98.89**	88.64	85.75	87.90	97.84
Magic04	82.12	76.12	72.69	77.78	**83.00**
Wave	**99.99**	77.91	66.80	78.27	78.50
RingCurve	**100.00**	99.26	90.11	99.38	99.98
LetRecog	**94.45**	74.20	64.01	74.31	88.81
Shuttle	99.75	92.67	85.66	94.48	**99.85**
OneBig	**99.99**	99.98	99.89	99.98	99.69
MiniBooNE	85.95	86.06	83.40	86.18	**89.58**
Mulcross	100.00	100.00	100.00	100.00	100.00
CoverType	**81.91**	66.72	63.05	66.57	72.89

Table XI: Win:Loss:Draw counts, as a result of a significant test based on two standard errors, for the classifier on the row versus the classifier on the column.

	AODE	BayesNet	NB-GD	NB-KDE
DEMass-Bayes	6:3:1	7:0:3	9:0:1	7:0:3
NB-KDE	1:7:2	0:2:8	9:0:1	
NB-GD	1:8:1	0:9:1		
BayesNet	1:7:2			

Bayesian clasifiers in three data sets - Magic04, Shuttle, and MiniBooNE.

A statistical test based on two standard errors was performed to examine whether the difference is significant. The win:loss:draw counts between each pair of classifiers are reported in Table XI. A win or loss is counted if the difference is significant; otherwise it is a draw. DEMass-Bayes had 6 wins, 3 losses and 1 draw when compared to AODE, a Bayesian classifier with the state-of-the-art performance. It had 7 wins, and 3 draws in comparison to NB-KDE and BayesNet.

Table XII shows that DEMass-Bayes was expensive in terms of run time. However, the increase in run time was not significant when compared to the existing Bayesian classifiers with increase in data size and the number of dimensions. We conduct a scale up test in the following subsection.

Table XII: Run time (in seconds) of a 10-fold cross validation for DEMass-Bayes and existing Bayesian classifiers: NB-KDE, NB-GD, BayesNet and AODE with default parameters.

Data set	DEMass -Bayes	NB-KDE	NB-GD	Bayes Net	AODE
Pendigits	2562	16	3	6	5
Magic04	423	93	4	9	5
Wave	237	24	2	4	3
RingCurve	227	22	3	4	4
LetRecog	4492	23	9	10	11
Shuttle	608	14	9	20	9
OneBig	3611	2361	28	101	32
MiniBooNE	3919	8594	134	375	119
Mulcross	1037	1832	29	64	20
CoverType	14088	1006	173	388	102

1) Scale up test: In order to examine how well the classifiers scale-up to large data size, we used the 48-dimensional Ring-Curve-Wave-TriGaussian data set, used in section VI-A. Data size was increased from 7000 to 70000, half-a-million, 1 million and 10 million. Figure 7 showed the increase in runtime of DEMass-Bayes and the existing Bayesian classifiers. With the increase in data size by a factor of 10, 75, and 150, DEMass-Bayes increased its runtime by a factor of 2, 9, and 17. The closest contender AODE increased its runtime by a factor of 6, 45, and 91, followed by NB-GD (12, 128, 286), BayesNet (15, 167, 374), and NB-KDE (38, 2345, 8721). Even with the data size increase by a factor of 1500, DEMass-Bayes only increased its runtime by a factor of 190, whereas BayesNet, NB-GD and AODE increased their runtime by factors of 7046, 6665 and 1038 respectively. DEMass-Bayes has a better scale up capability than the existing Bayesian classifiers.

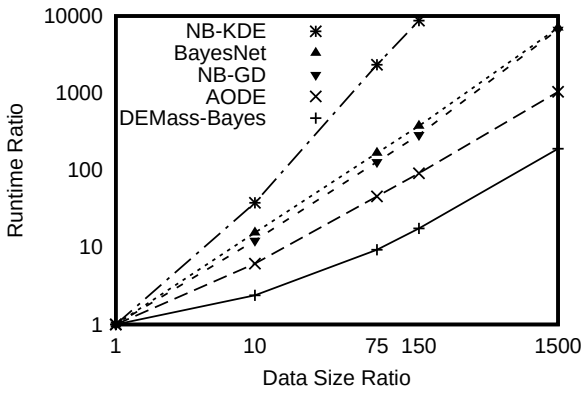


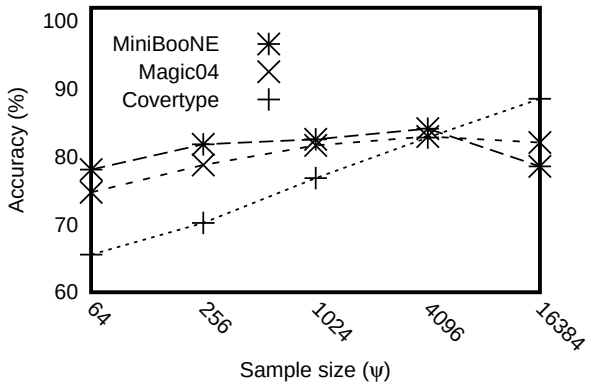
Figure 7: Scale up: DEMass-Bayes versus existing Bayesian Classifiers in the 48-dimensional Ring-Curve-Wave-TriGaussian data set. The base for data size ratio is 7000 instances and the base for runtime ratio is the runtime on 7000 instances. Axes are on logarithmic scale of base 10.

2) Sensitivity of parameters: In order to examine the effect of two parameters, sample size ψ and number of trees t on the classification accuracy of DEMass-Bayes, we ran two experiments on 6 real data sets, namely Pendigits, Shuttle, Letter Recognition, Magic04, MiniBooNE and Covertype:

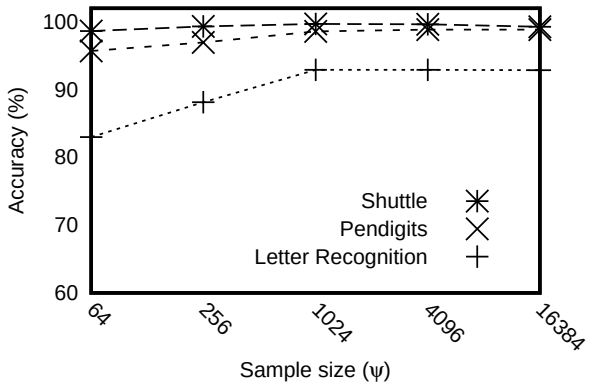
1) Vary sample size ψ with a fixed number of trees, t =100.
2) Vary number of trees t with a fixed sample size, ψ =256.

Figures 8 and 9 showed the results of these two expriments respectively.

With increase in sample size ψ, the accuracy increased up to a certain point and then remained flat except in MiniBooNE, where the accuracy decreased as the sample

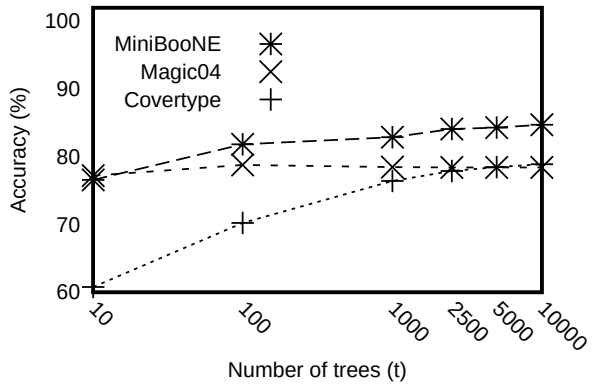


(a) Effect of ψ in Magic04, MiniBooNE, and Covertype data sets

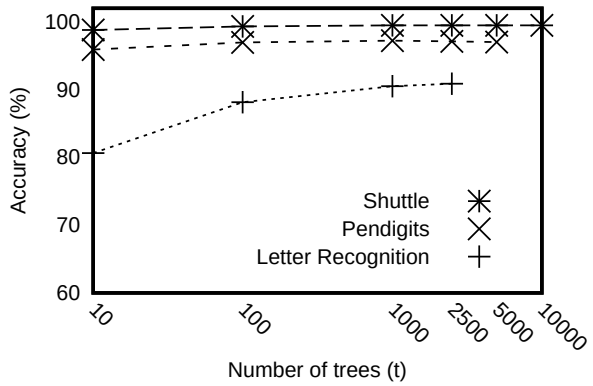


(b) Effect of ψ in Pendigits, Shuttle, and Letter Recognition data sets

Figure 8: Effect of sample size (ψ) on the accuracy of DEMass-Bayes (t=100) in real data sets. Horizontal axis is on logarithmic scale of base 2. Accuracies were measured on ψ=64, 256, 1024, 4096, and 16384.



(a) Effect of t in Magic04, MiniBooNE, and Covertype data sets



(b) Effect of t in Pendigits, Shuttle, and Letter Recognition data sets

Figure 9: Effect of number of trees (t) on the accuracy of DEMass-bayes (ψ=256) in real data sets. Horizontal axis is on logarithmic scale of base 10. Accuracies were measured on t=10, 100, 1000, 2500, 5000, and 10000. We could not run DEMass-Bayes on Letter Recognition with t =5000, and 10000 and on Pendigits with t =10000 due to insufficient memory.

size was increased from 4096. But, in case of Covertype, accuracy kept increasing because the two biggest classes have more than two hundred fifty thousand instances each and the continuous accuracy improvements is a result of improved accuracy for these two classes.

When the number of trees t was increased, the accuracy increased initially and remained constant after reaching a certain point.

VII. Discussion

What we have presented is the first density estimation method that utilizes no distance measures. It potentially solves fundamental problems such as the curse of dimensionality in which the use of a distance measure plays a key part in creating the problem [3], [10].

There are significant improvements of nearest neighbour search in recent times. For example, indexing schemes to speed up nearest neighbour search such as Cover Trees [4] and M-Trees [6] are claimed to have time complexity significantly better than $O(n^2)$. Indexing schemes such as Cover Trees or M-Trees rely on distance-based pruning methods in both the index tree construction and range query processes. Distance-based pruning methods cannot scale up to massive data, and they are known to be inefficient even for a moderate number of dimensions. Thus, it is unlikely that any of the recent indexing schemes can be used to speed up nearest neighbour search to the level that has been achieved already by DEMass-DBSCAN and DEMass-LOF, especially in large data sets.

Note that the purpose of trees used in DEMass differs from that used for Cover Trees or M-Trees. Trees in DEMass are used to estimate mass, the core computation process. In contrast, Cover Trees or M-Trees are indices used to speed up nearest neighbour search. The indices are required because the core computation, i.e., the requirement to calculate distance for every pair of instances, is slow. In other words,

one uses trees directly in the core process; and the other uses trees to aid the core process where trees are not used in the actual computation of distance.

The cost of KDE estimation can be lowered, for example, by reducing the given data set D to some 'representative' subset, where each representative kernel is derived from a subsample using a maximum likelihood method such as EM. This reduces the KDE estimation time; but it comes with a cost of an expensive pre-processing step.

It is possible to use neighbours to compute LOF for DEMass-LOF. However, the runtime advantage over LOF will be significantly reduced because of the additional computations required to calculate the density of each neighbour, even though it does not need to find neighbours based on distance calculations.

DENCLUE [11], a generic density-based algorithm, builds a density distribution from data, and then uses a threshold to determine clusters—all connected points above the threshold form a cluster. DBSCAN is a special case of DENCLUE. DEMass-DENCLUE has exactly the same procedure as DEMass-DBSCAN, where $Minpts$ or the equivalent density threshold stated in Section V-A is employed as the threshold.

DEMass sets a new benchmark of what density-based algorithms can achieve. In contrast to the density-based approaches, mass-based approaches [21], [20] solve problems without the use of a density estimator. Mass-based approaches have been shown to perform better than the current density-based approaches in terms of time and space complexities. It is thus interesting to compare the new benchmark achieved by DEMass-density-based approaches with mass-based approaches.

The current implementation of DEMass has two limitations. First, it has step subdivisions controlled by a global parameter h. The limited possible steps may be too coarse for some applications and the setting is not adaptive to local variations in density. Second, the grid-based implementation carries all the limitations associated with grid-based approaches, especially dealing high dimensional problems. All these limitations can be overcome by using a non-grid method which is adaptive to the local data distribution. This non-grid-based implementation will eliminate one global parameter and potentially tackle high-dimensional problems more effectively.

VIII. Conclusions and Future Work

The new density estimation method we introduced have two unique features which cannot be found in existing density estimation methods. First, it is the first density estimator that utilizes no distance measures. Second, it has average case sublinear time complexity and constant space complexity. Existing density estimators must use a distance measure and have time and space complexities a lot worse than linear. The time and space complexities achieved set a new benchmark for density-based algorithms, of what previously thought impossible.

The bias-variance analysis reveals that the new density estimator has the same characteristic as kernel density estimator, i.e., both have a smoothing parameter used to trade-off between systematic error (bias) and random error (variance).

Making full use of the features in the new density estimator, we show that two current algorithms, in the unsupervised learning setting from two key areas of data mining, can be significantly simplified through set-based definitions rather than the current point-based definitions. This has directly contributed to their improved time complexities. In the supervised learning setting, DEMass enables direct estimation of $p(\mathbf{x}|y)$ for the first time, without any assumption.

Our evaluation shows that the new density estimator not only successfully replaces existing density estimators in three density-based algorithms, DBSCAN, LOF and Bayesian classifiers, but reduces their runtime to become algorithms with the lowest sub-linear time complexity. In addition, DEMass-DBSCAN, DEMass-LOF and DEMass-Bayes often achieve equivalent or better task-specific performances than DBSCAN, LOF and existing Bayesian classifiers.

Our result implies that most, if not all, density-based algorithms can reap the immediate benefit of significantly lowering their time complexities by simply replacing the existing density estimators with the new one, with a potential further improvement in the task-specific performance.

Future work has three directions. First, we will apply the new density estimator in existing algorithms in more areas. We will ascertain whether there are areas in which the new density estimator cannot replace existing density estimators. Second, compare DEMass-density-based approaches with mass-based approaches to determine their relative strengths and weaknesses. Third, we will explore DEMass's ability to deal with high dimensional problems.

IX. Acknowledgments

This work is partially supported by the Air Force Research Laboratory, under agreement# FA2386-10-1-4052. Takashi Washio is partially supported by JSPS Grant-in-Aid for Scientific Research(B)# 22300054. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Xiao Yu Ge assisted in experiments using ELKI. Hiroshi Motoda, Zhouyu Fu, and the anonymous reviewers had provided many helpful comments to improve this paper.

The source code of DEMass-DBSCAN is available at http://sourceforge.net/projects/mass-estimation/.

References

[1] E. Achtert, H.-P. Kriegel, and A. Zimek. Elki: A software system for evaluation of subspace clustering algorithms. In *Proceedings of the 20th International Conference on Scientific ans Statistical Database Management*, pages 580–585, 2008.

[2] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[3] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory*, pages 217–235, 1999.

[4] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 97–104, 2006.

[5] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying density-based local outliers. In *Proceedings of ACM SIGMOD*, pages 93–104, 2000.

[6] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 426–435, 1997.

[7] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of KDD*, pages 226–231, 1996.

[8] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous valued attributes for classification learning. In *Proceedings of 14th International Joint Conference on Artificial Intelligence*, pages 1034–1040, 1995.

[9] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29:131–163, 1997.

[10] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What is the nearest neighbor in high dimensional spaces? In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 506–515, 2000.

[11] A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proceedings of KDD*, pages 58–65. AAAI Press, 1998.

[12] P. Langley, W. Iba, and K. Thompson. An analysis of bayesian classifiers. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 399–406, 1992.

[13] P. Langley and G. H. John. Estimating continuous distribution in bayesian classifiers. In *Proceedings of Eleventh conference on uncertainty in artificial intelligence*, 1995.

[14] F. T. Liu, K. M. Ting, and Z.-H. Zhou. On detecting clustered anomalies using sciforest. In *Proceedings of ECML PKDD*, pages 274–290, 2010.

[15] A. Nanopoulos, Y. Theodoridis, and Y. Manolopoulos. Indexed-based density biased sampling for clustering applications. *IEEE Transaction on Data and Knowledge Engineering*, 57(1):37–63, 2006.

[16] D. M. Rocke and D. L. Woodruff. Identification of outliers in multivariate data. *Journal of the American Statistical Association*, 91(435):1047–1061, 1996.

[17] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapmal & Hall/CRC, 1986.

[18] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2006.

[19] S. C. Tan, K. M. Ting, and F. T. Liu. Fast anomaly detection for streaming data. In *Proceedings of IJCAI.*, pages 1151–1156, 2011.

[20] K. M. Ting and J. R. Wells. Multi-dimensional mass estimation and mass-based clustering. In *Proceedings of IEEE ICDM*, pages 511–520, 2010.

[21] K. M. Ting, G.-T. Zhou, F. T. Liu, and S. C. Tan. Mass estimation and its applications. In *Proceedings of ACM SIGKDD*, pages 989–998, 2010.

[22] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Second Edition. Springer, 2000.

[23] G. I. Webb, J. R. Boughton, and Z. Wang. Aggregating one-dependence estimators. *Machine Learning*, 58:5–24, 2005.

[24] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Second Edition. Morgan Kaufmann, 2005.

[25] K. Yamanishi, J.-I. Takeuchi, G. Williams, and P. Milne. Online unsupervised outlier detection using finite mixtures with discounting learning algorithms. In *Proceedings of ACM SIGKDD*, pages 320–324, 2000.

APPENDIX - DATA CHARACTERISTIC

The characteristic of the data set, Ring-Curve-Wave-TriGaussian, used in Section V-A is shown in Figure 10. Each of the Ring-Curve, Wave and Triangular-Gaussian is a two-dimensional data set; and together there is a total of seven clusters. Each cluster has 10000 instances. When used in the scale up experiment, the data size in each cluster was scaled by a factor of 0.1, 1, 75, 150 to 1500.

(a) Ring-Curve (b) Wave (c) Tri-Gaussian

Figure 10: Scatter plot of the clusters in the Ring-Curve-Wave-TriGaussian data set, as used in [20].

Table 17 AUC values for anomaly detection, comparing `MassAD` with $\texttt{DepthAD}_s$ (which employed either data depth or local data depth) that build a single model from the entire data set.

	`MassAD`		$\texttt{DepthAD}_s$	
	$\texttt{Mass}''$	$\texttt{Mass}'$	`Depth`	`LDepth`
Http	**1.00**	**1.00**	0.84	0.50
Forest	0.90	**0.92**	0.50	0.55
Mulcross	0.26	**0.99**	0.88	0.61
Smtp	**0.91**	0.86	0.86	0.76
Shuttle	**1.00**	0.99	0.51	0.70
Mammography	**0.86**	0.37	0.73	0.62
Annthyroid	0.75	0.71	0.59	**0.85**
Satellite	**0.77**	0.62	0.50	0.70